POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

DÉCOUPAGE VIRTUEL INTERACTIF DE CORPS ÉLASTIQUES POUR SIMULATION CHIRURGICALE

VINCENT MAGNOUX
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Juillet 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

DÉCOUPAGE VIRTUEL INTERACTIF DE CORPS ÉLASTIQUES POUR SIMULATION CHIRURGICALE

présentée par **Vincent MAGNOUX**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Michel DAGENAIS, président
Benoît OZELL, membre et directeur de recherche
André GARON, membre
Guillaume HOUZEAUX, membre externe

DÉDICACE

À Zygotte
et Matière Blanche

REMERCIEMENTS

Je tiens à remercier tout spécialement Benoît Ozell pour son aide apportée tout au long de mon travail. Merci d'avoir été si patient avec moi, pour ta disponibilité, tes réponses rapides, ton soutien, tes encouragements, tes conseils, ton temps passé à discuter de mes difficultés (et des solutions !), pour m'avoir guidé et m'avoir aidé à garder l'intérêt. Bref, pour m'avoir si bien accompagné durant cette période. Ça a été toute une étape et je t'en suis très reconnaissant :)

RÉSUMÉ

La simulation chirurgicale dans un environnement de réalité virtuelle fournit un moyen de pratiquer certaines opérations sans les risques associés à une intervention sur un patient ou le coût d'un mannequin. Afin de générer un sentiment de présence, on cherche à produire un environnement le plus complet possible, incluant une vision 3D, un retour haptique, des interactions crédibles et un comportement physique réaliste des objets présents.

La recherche présentée ici porte sur la simulation physique du comportement d'organes mous, comme le foie ou le cerveau, ainsi que sur le découpage de ces organes à l'aide d'un scalpel, une interaction particulièrement difficile à reproduire virtuellement de façon réaliste. L'objectif principal est de développer une méthode de déformation à la fois réaliste et efficace, et de permettre à un utilisateur de découper interactivement un objet simulé par cette méthode, à l'aide d'un outil tranchant virtuel. De plus, nous voulons que la déformation et les interactions soient décrites avec une grande précision, tout en permettant d'effectuer les calculs très rapidement, pour une interaction fluide qui maintient le sentiment de présence.

Pour reproduire fidèlement le comportement physique des organes, nous utilisons une méthode basée sur la mécanique des milieux continus. Une des méthodes les plus utilisées pour ce genre de simulation est celle des éléments finis. Cependant, étant donné qu'elle dépend d'une division, ou discrétisation, des objets simulés en un maillage tridimensionnel avec une structure rigide, il est difficile de découper ces objets, puisqu'il faut alors redéfinir la discrétisation. Pour faciliter le découpage, nous choisissons plutôt une méthode dite par particules, qui définit de façon moins rigide les liens entre les parties discrètes de l'objet simulé.

La première étape de ce projet consiste donc à définir une description du lien entre les particules qui forment un objet et d'une méthode pour reforger ces liens lorsqu'un outil découpe l'objet. Pour bien visualiser l'objet, nous utilisons une surface explicite formée de triangles. Nous développons également une méthode pour modifier progressivement cette surface d'après la position d'un outil tranchant, de concert avec les modifications apportées aux liens entre particules. Nous démontrons le réalisme de cette méthode en la comparant avec celle des éléments finis, et nous démontrons que le découpage demande une faible quantité de ressources par rapport au calcul de la déformation.

Ensuite, afin de rendre le découpage interactif, nous développons une description du mouvement du fil de la lame d'un outil et combinons ce mouvement avec la méthode de découpage et celle de déformation établie précédemment. Une attention particulière est portée à cette intégration pour que le mouvement et la déformation provoqués par l'ajout d'une coupure ne

causent pas d'intersections supplémentaires avec la lame. En ce qui concerne le mappage de la surface sur les particules, nous développons un ensemble de critères pour associer les sommets des triangles avec les bonnes particules au fur et à mesure qu'une coupure est produite. Nous démontrons le bon fonctionnement de ces méthodes dans des situations diverses, mais sans nécessairement atteindre une vitesse de rafraîchissement en temps réel.

Enfin, pour obtenir une simulation à grande résolution en temps réel, nous développons une structure de données qui permet de décrire les liens entre particules d'une façon utilisable par une carte graphique (GPU). L'utilisation du GPU pour effectuer les calculs de déformation – l'aspect le plus coûteux de la simulation – demande en effet une organisation particulière des données pour un accès efficace. Cependant, étant donné que les liens entre particules sont modifiés constamment, il est également nécessaire que cette structure soit mise à jour de façon efficace. Nous démontrons la performance de l'implantation GPU en la comparant à l'implantation de base sur des objets simulés avec un grand nombre de particules.

L'ensemble de ces travaux ont permis de remplir les objectifs posés de déformer et découper interactivement des corps déformables. Cependant, des ajouts sont encore nécessaires pour que la méthode développée puisse être utilisée dans un simulateur plus complet. Nous proposons donc des méthodes pour améliorer la stabilité de la simulation, du point de vue visuel et physique, et pour lui ajouter le calcul d'un retour d'effort pour l'utilisateur.

ABSTRACT

Surgery simulation in a virtual reality environment provides a way to practice certain operations without the risks associated with performing surgery on a patient or the cost of using a realistic dummy. To facilitate immersion, we seek to produce an environment as complete as possible, including 3D vision, haptic feedback, credible interactions and a realistic physical behavior of simulated objects.

The research presented in this document focuses on the physical behavior of soft organs, like the brain or liver, and on cutting these organs using a scalpel. It is especially difficult to reproduce virtually that interaction in a realistic way. The main objective is to develop a deformation method that is both realistic and efficient, and to allow a user to interactively cut an object simulated through this method, using a virtual sharp tool. Furthermore, we want the deformation and interactions to be described with high precision while allowing for fast computations, for a smooth interaction that maintains immersion.

For the first step, we define a description of the connections between the particles that form an object and a method to modify these connections when a tool cuts the object. In order to provide a realistic visual rendering, we use an explicit surface made of triangles. We also develop a method to progressively modify this surface from the movement of a sharp tool, while the connections between particles are being updated. We demonstrate the accuracy of this method by comparing it to a finite element simulation, and we show that cutting only requires a small amount of resources compared to deformation computation.

We then develop a description of the movement of the edge of a tool and combine this movement with the cutting and deformation method established earlier, to make the cutting operation interactive. This combination must be done carefully so that the movement and deformation resulting from a cut do not cause additional intersections with the tool edge. For mapping the surface onto the particles, we develop a small set of criteria that associate triangle vertices with the right particles while a cut is produced. We demonstrate these methods in various scenarios, without necessarily reaching a real time update rate.

Finally, to obtain a high resolution, real time interaction, we develop a data structure that describes the connections between particles in a way that is suitable for graphics card (GPU) computations. The use of a GPU to perform deformation computations, the costliest aspect of the simulation, requires the data to be structured in a specific way so that it can be accessed in an efficient way. However, since the connections are constantly changing, we must also be able to update this data structure efficiently. We demonstrate the performance

of the GPU implementation of our method by comparing it to the base implementation on objects represented with a large number of particles.

This work has fulfilled the initial objectives of interactively deforming and cutting elastic bodies. However, to use the developed method in a more complete simulator, some additional features are required. We thus propose methods to improve the stability of our simulation, physically and visually, and to add haptic feedback computations for the user.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xiii

LISTE DES FIGURES xiv

LISTE DES SIGLES ET ABRÉVIATIONS xix

LISTE DES ANNEXES xx

CHAPITRE 1 INTRODUCTION 1
1.1 Présentation du problème 2
1.1.1 Déformation 2
1.1.2 Interaction 2
1.1.3 Découpage 2
1.1.4 Vitesse d'exécution 3
1.1.5 Plan de la thèse 3

CHAPITRE 2 REVUE DE LITTÉRATURE 4
2.1 Déformation 4
2.1.1 Éléments finis avec remaillage 7
2.1.2 Approches multirésolutions 8
2.1.3 Éléments finis étendus (XFEM) 9
2.1.4 Éléments finis de frontière 10
2.1.5 Particules 10
2.1.6 Méthode basée sur les nœuds (PBA) 12
2.1.7 Particules avec grille d'intégration 15
2.1.8 Collocation 17

2.1.9 Dynamique basée sur la position 18
2.1.10 Masse-ressort 21
2.2 Solveurs 21
2.2.1 Statique 22
2.2.2 Explicite 22
2.2.3 Implicite 23
2.2.4 Mixte 24
2.3 Rendu visuel 24
2.4 Rendu haptique 25
2.5 Objectifs de recherche 27
2.5.1 Hypothèses 27
2.5.2 Objectifs spécifiques 27

CHAPITRE 3 MÉTHODE 28
3.1 Contexte du projet 28
3.1.1 Déformation physiquement réaliste 28
3.1.2 Coupe libre, exacte et progressive 28
3.1.3 Faible coût de la simulation des coupures 28
3.1.4 Interactivité 29
3.1.5 Implantation GPU 29
3.2 Environnement 29
3.2.1 Simulateur chirurgical 29
3.2.2 Développement 30
3.3 Axes de recherche 30
3.3.1 Real-time visual and physical cutting of a meshless model deformed on a background grid 31
3.3.2 Dynamic cutting of a meshless model for interactive surgery simulation 31
3.3.3 GPU-friendly data structures for real time simulation 31

CHAPITRE 4 REAL-TIME VISUAL AND PHYSICAL CUTTING OF A MESHLESS MODEL DEFORMED ON A BACKGROUND GRID 32
4.1 Introduction 32
4.1.1 Related Work 33
4.1.2 Contributions 36
4.2 Method 36
4.2.1 Deformation Model 37
4.2.2 Discretization 37

4.2.3 Object Topology . . . 39
4.2.4 Surface . . . 40
4.3 Results . . . 44
4.3.1 Versatility and Performance . . . 44
4.3.2 Accuracy . . . 48
4.3.3 Limitations . . . 50
4.4 Conclusion and Future Work . . . 51

CHAPITRE 5 DYNAMIC CUTTING OF A MESHLESS MODEL FOR INTERACTIVE SURGERY SIMULATION . . . 52
5.1 Introduction . . . 52
5.1.1 Background and Related Work . . . 53
5.1.2 Contributions . . . 55
5.2 Method . . . 55
5.2.1 Tool Representation and Collision Detection . . . 55
5.2.2 Surface Mapping and Remapping . . . 58
5.2.3 Dynamic Simulation . . . 60
5.3 Results . . . 62
5.4 Conclusion and Future Work . . . 66

CHAPITRE 6 GPU-FRIENDLY DATA STRUCTURES FOR REAL TIME SIMULATION . . . 67
6.1 Introduction . . . 67
6.1.1 Background . . . 68
6.1.2 Contributions . . . 70
6.2 Method . . . 70
6.2.1 Deformation . . . 71
6.2.2 Changing connectivity data . . . 73
6.3 Results . . . 76
6.3.1 Performance Comparisons . . . 76
6.3.2 Limitations . . . 79
6.4 Conclusion and Future Work . . . 80

CHAPITRE 7 DISCUSSION GÉNÉRALE . . . 83
7.1 Travaux complémentaires . . . 83
7.1.1 Stabilité . . . 83
7.1.2 Mappage de surface . . . 83

7.1.3 Détection de collisions 84
7.1.4 Retour haptique 85
7.2 Retour sur les objectifs 85
7.2.1 Implantation d'une méthode avec particules et découpage 85
7.2.2 Gestion de la surface 86
7.2.3 Utilisation du GPU 86
7.3 Résultats des travaux de recherche 86
7.4 Contributions 87

CHAPITRE 8 CONCLUSION 88
8.1 Synthèse des travaux 88
8.2 Limitations 88
8.3 Améliorations futures 89

RÉFÉRENCES 90

ANNEXES 104

LISTE DES TABLEAUX

4.1 Error in the displacement of the tip of the beam in the tested scenarios with respect to the FEM model used as a reference. The error is expressed in percentage relative to the total displacement of the reference solution (not to the dimension of the model). The two error columns show the result for a simulation with 8 or 4 neighbors per integration point, and one where integration element volumes are not estimated ("One volume"). 50

6.1 Summary of method and reported performance for other GPU-based surgery simulators, compared to our method. 82

LISTE DES FIGURES

4.1 Two-dimensional representation of the grids used to compute deformation. a) The background integration grid; each square is an integration element, with a corresponding integration point in its center (blue crosses). b) The degrees of freedom (green dots) of the system are placed in a larger grid that coincides with the background grid. In 2D, each group of four DOFs form a cell (green squares). c) The volume (or surface when in 2D) of the integration elements is determined by the proportion that is covered by the object; in this particular example geometry, the two bottom corner integration points (in orange) are discarded. 38

4.2 Choosing neighbors during volume cutting. Solid lines indicate links between two DOFs and dashed lines indicate links between a DOF and an integration point. a) Initially, neighbors are chosen only by proximity. b) One particle is cut off integration point A; a new neighbor B is chosen for that point by following the links between particles (in purple) and selecting the closest one which does not result in a collinear set of neighbors. c) The third neighbor is cut off the integration point; since the next topologically closest particle (C) makes the neighborhood collinear, we insert an additional neighbor D to be able to generate valid shape functions. 40

4.3 The process of cutting the existing surface. a) The initial state of the surface. b) The tool intersected edges A, B and C, generating three new pairs of vertices (blue squares), and stopped inside a triangle, generating a single additional vertex (red square). c) The tool progressed through the surface, but without intersecting any edge and at an angle slightly different from the previous frame; the vertex on the edge of the blade is simply displaced; the model has also been deformed since the previous frame. d) The tool intersected edges D and E, generating two new pairs of vertices and displacing the one on the edge. 41

4.4 Meshing the surface revealed by cutting, using a moving cutting front. Circles represent edges that may be intersected by the cut. The red squares are the vertices composing the cutting front. The gray rectangle is the cutting plane formed by the progression of the cutting tool (from left to right). a) Initial surface. b) Cut starting from the left. The ends of the cutting front are determined by the intersections between the blade and the existing surface. An additional transient point A is placed in the middle of the front. c) The blade progresses, but the generated triangles are still small enough that no new ones are generated, the point A is simply moved forward. d) When moving forward, transient point A was too far from the nearest vertices, so it left a new fixed point C behind; since the front has become too wide, point A has been split in two (generating a new transient point B). 42

4.5 a) Multiple cuts were made in a torus. b) A section of the torus was completely separated from the rest of the body. 44

4.6 Detail of the triangles being generated as the cutting tool progresses through the object's initial surface. a) A torus with a coarse surface. b) A liver with a high-resolution surface. c) An object with a star-shaped cross-section. 45

4.7 Evolution of total simulation time per frame with respect to surface mesh size. Total time consists of deformation, collision detection and cutting time. The different lines correspond to different volumetric model sizes, in number of elements. 46

4.8 Evolution of the time taken to perform all topology change operations, with respect to surface mesh size. The different lines correspond to different volumetric model sizes, in number of elements. 47

4.9 Percentage of frame time used for cutting, evolving with respect to the number of triangles in the surface mesh. Individual lines correspond to different resolutions of the volumetric model, in number of integration elements. 48

4.10 A high-resolution surface is mapped to an underlying physical model with lower resolution (DOFs are shown as green dots). The model has been cut and deformed. 49

4.11 Surface model of the cylinder used for testing accuracy. a) Initial model. b) Cut model, after deformation. 49

5.1 The area swept by the cutting tool is only described in terms of the position of its edge at the previous frame (t_0) and its position at the current frame (t_1). 56
5.2 Example of *Situation 1*, in 2D, where a collision detection scheme that is continuous only with the tool movement would fail to detect a certain triangle edge. The large arrows indicate the direction of movement of the tool and object. The red line between tool positions represents the space swept by it between frames. 57
5.3 Example of *Situation 2*, where the line spanned by the tip of the cutting tool intersects two of the objects edges at the end of the frame, even though the tool was outside the object at every point in time between the two frames. The large arrows indicate the direction of movement of the tool and object. The red line between tool positions represents the space swept by it between frames. 57
5.4 Example of *Situation 3*, where a simulated object moves downward onto a motionless cutting tool. The tool's edge does not sweep any area, but must still be able to cut the object. 58
5.5 Illustration in 2D of a situation where the nearest integration point to a certain vertex would not be the best mapping. Integration points are displayed as red crosses and vertices as blue circles, with dotted lines indicating the nearest integration point. Vertex A has integration point 1 as its closest, which lies across another surface boundary. We would rather choose an integration point that is *not* separated by another part of the surface, such as point 2, connected to the thicker green line. 60
5.6 Illustration of the need for an additional constraint plane when finding a good mapping. *Left:* A cutting tool approaches from the left, below vertex A. *Middle:* The introduced cut has caused vertices A and B to lose their previous mapping. Their initial normal is indicated. Using only that normal would cause them to be mapped on the integration point 1, from which they were just unmapped. *Right:* The introduced cutting plane causes vertices A and B to be mapped to point 2. . . . 61
5.7 A circular cylinder is moving downward and being cut by a horizontal edge that remains in the same place. The surface of the cylinder (including the newly-generated part) properly follows the movement of the underlying physical model. 63

5.8 A horizontal cutting edge moves upward in an fixed circular cylinder and cuts it. 63
5.9 Cutting edge (in black) that is curved and that twists as it moves forward. 64
5.10 Cutting a more complex model that may appear in a surgical simulation scenario. The cutting tool is able to deal with the numerous folds present in a human brain. 64
5.11 Illustration of a series of cuts in the same object. A slight curve is noticeable in the first cut and is caused by the movement of the liver under the effect of gravity while the cut was performed. 65
5.12 A view of the liver cut scenario that displays the integration points (red crosses) and highlights to which point each vertex is mapped (blue lines). Each vertex is properly mapped to a point on its own side of the cut. 65
6.1 Illustration of what the four large 2D data arrays needed for updating an object's state represent. Red crosses are the center of elements, green circles the simulation nodes and blue square are surface vertices. The connectivity links elements to nodes, with shape function values and derivatives associated with each pair. The surface mapping links vertices to nodes, with a weight associated with each pair. 74
6.2 Execution steps and data flow of the main simulation loop highlighting which operations are performed on the CPU and on the GPU. Boldface data are the large 2D structures described in this paper. 75
6.3 Data structure used for major 2D arrays – connectivity data is illustrated here. In the first, large array, rows all occupy the same amount of memory, regardless of the number of elements in them. A secondary array contains the number of elements (length) in every row. 76
6.4 Total update time (in ms) for different configurations, with respect to number of elements in physical model. 77
6.5 Total update time as a function of the number of triangles in the surface mesh. The volumetric model used for these tests has 20k elements, which was the approximate size limit to obtain an interactive update rate on the CPU. 78
6.6 Proportion of execution time taken by each major step of the main simulation loop, for the case using single precision floating points on the GPU, with 20k volume elements. 79

6.7 Comparison of total update times for the GPU version of the simulation with and without splitting 2D arrays of data into batches, both in single and double precision. 80
6.8 Proportion of execution time taken by different aspects of the deformation step, using single precision. Computations take most of the time, mainly on the GPU, while memory transfer and CUDA overhead remain low. 81

LISTE DES SIGLES ET ABRÉVIATIONS

CFE	Éléments finis composites (Composite Finite Elements)
CPU	Processeur central (Central Processing Unit)
DOF	Degré de liberté (Degree Of Freedom)
EF	Éléments finis
EFG	Méthode Galerkin sans éléments (Element Free Galerkin)
FEM	Méthode des éléments finis (Finite Element Method)
GPU	Processeur graphique (Graphics Processing Unit)
LCP	Problème de complémentarité linéaire (Linear Complementarity Problem)
MLS	Moindres carrés mobiles (Moving Least Squares)
PBA	Animation basée sur les points (Point Based Animation)
PBD	Dynamique basée sur la position (Position Based Dynamics)
PIM	Méthode d'interpolation sur les points (Point Interpolation Method)
RBF	Fonction de base radiale (Radial Basis Function)
SPH	Smoothed Particle Hydrodynamics
XFEM	Méthode des éléments finis étendue (Extended Finite Element Method)

LISTE DES ANNEXES

Annexe A Approximation MLS 104
Annexe B Solveur implicite 107
Annexe C Résolution de contact 109

CHAPITRE 1 INTRODUCTION

La simulation offre une solution à plusieurs problèmes reliés à la formation et l'entraînement de chirurgiens. Elle fournit notamment un environnement permettant d'apprendre les gestes nécessaires pour une procédure sans les risques d'une intervention chirurgicale réelle. C'est également un moyen plus flexible et moins coûteux que l'entraînement sur des modèles artificiels, des cadavres ou sur des animaux, en plus d'éviter les problèmes éthiques rattachés à cette dernière solution.

L'utilité d'une simulation chirurgicale provient de son réalisme, de l'applicabilité des gestes appris durant la simulation à une salle d'opération réelle, sur un patient vivant. Le réalisme repose sur l'apparence de l'environnement simulé et de son comportement face aux manipulations de l'utilisateur. Étant donné que le corps humain est composé à la fois de tissus rigides (os), déformables (organes, ligaments) et liquides (sang), ce comportement peut être très complexe à reproduire fidèlement.

Les simulateurs existants comportent généralement un appareil permettant de transmettre les mouvements de l'utilisateur à l'environnement virtuel et, à l'inverse, de retourner une sensation de toucher de l'environnement à l'utilisateur, appelé retour ou rendu haptique. Ce rendu haptique contribue de façon importante au réalisme d'une simulation.

Ainsi, au-delà de la difficulté liée à la modélisation du comportement physique de l'environnement simulé, des critères techniques stricts viennent s'ajouter. Afin de créer un sentiment de présence pour l'utilisateur, le résultat de la simulation doit lui être transmis en temps réel – tant du point de vue visuel que haptique – et ne doit jamais soudainement s'éloigner du comportement modélisé. Spécifiquement, on désire remplir des critères de vitesse, stabilité, rendu visuel et rendu haptique.

Avec la recherche présentée dans ce document, nous visons à développer un moteur régi par des lois physiques qui permettra la simulation interactive de déformation de corps mous ainsi que de résection d'organes ou de tissu pathologique. La modélisation du comportement physique doit être aussi précise que possible, et la représentation des coupures doit suivre exactement les mouvements de l'outil sous le contrôle de l'utilisateur.

1.1 Présentation du problème

1.1.1 Déformation

Nous cherchons à modéliser le plus fidèlement possible le comportement des organes lors d'une opération chirurgicale. Ce problème consiste à déterminer la position et la conformation d'un organe en tout temps, selon les forces externes qui lui sont appliquées – par exemple la gravité ou le contact avec un outil chirurgical – et selon les forces élastiques internes causées par sa déformation. Il peut être décrit par l'équation suivante :

$$\rho \frac{\partial^2 \boldsymbol{x}}{\partial t^2} - \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}, t), \tag{1.1}$$

où $\boldsymbol{x}$ correspond au champ décrivant la position de chaque point de l'organe, $\boldsymbol{\sigma}$ au tenseur de contraintes, ρ à la densité de masse et $\boldsymbol{f}$ à l'ensemble des forces externes. La variation temporelle des forces internes et les forces de viscosité sont ignorées ici ; l'équation 1.1 représente donc un corps hyperélastique. Étant donné la difficulté pour un ordinateur de représenter et résoudre des équations continues, ce problème doit être résolu en discrétisant le milieu simulé dans l'espace et dans le temps. La façon d'effectuer cette discrétisation donne lieu aux différentes méthodes présentées dans ce document.

1.1.2 Interaction

Le seul moyen par lequel l'utilisateur pourra interagir directement avec la scène simulée sera en contrôlant un outil virtuel, par exemple à l'aide d'un dispositif haptique. Il faut donc également simuler la présence d'un outil chirurgical rigide, ainsi que son interaction avec les organes déformables, et calculer les forces de contact induites par cette interaction. Par ailleurs, afin que l'environnement simulé soit le plus réaliste possible, tous les éléments de la scène seront représentés visuellement. En ce qui concerne les corps déformables, ce rendu visuel dépend en partie de la méthode de discrétisation utilisée. La principale difficulté qui lui est associée vient cependant des changements de topologie sur le modèle physique de l'organe, qui doivent être propagés à son modèle visuel.

1.1.3 Découpage

Lorsque la topologie d'un organe est modifiée, sous l'effet d'une incision ou d'une déchirure par exemple, la discrétisation du modèle doit également être modifiée, ce qui peut affecter de façon importante la résolution du problème de déformation. De plus, la formation de la coupure doit provenir d'une interaction avec l'outil manipulé par l'utilisateur et doit s'insérer

dans la boucle d'animation, parmi les calculs de retour d'effort, de déformation et de détection des collisions. La complexité du positionnement du découpage au sein de l'animation provient de l'interdépendance de tous ces calculs.

1.1.4 Vitesse d'exécution

Pour que l'utilisateur puisse agir sur l'environnement virtuel et percevoir le résultat de ses actions de façon crédible, il est nécessaire d'effectuer tous les calculs à une vitesse interactive. De façon générale, des calculs plus efficaces de certains aspects de la simulation permettent d'améliorer ces aspects – par exemple en augmentant la résolution des organes modélisés – ou d'ajouter des éléments qui rehaussent le réalisme de la simulation – par exemple un rendu visuel plus complexe ou des interactions avec du sang. Une attention particulière est donc portée à la performance des méthodes discutées tout au long de ce projet. L'utilisation de carte graphique (GPU) pour effectuer les calculs permettra une amélioration supplémentaire de la performance de la simulation.

1.1.5 Plan de la thèse

Le chapitre 2 présente une revue de littérature et les objectifs de la thèse, tandis que le chapitre 3 présente un survol de la méthode utilisée pour atteindre ces objectifs. Les chapitres suivants décrivent le travail effectué sous la forme d'articles scientifiques, divisé en trois aspects : l'élaboration d'une méthode de découpage efficace et réaliste (chapitre 4), l'intégration de cette méthode dans une boucle d'animation pour la rendre interactive (chapitre 5) et le développement d'une structure de données permettant d'exécuter efficacement cette méthode sur du matériel graphique (chapitre 6). Le chapitre 7 discute ensuite de l'ensemble des résultats.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Déformation

Le problème du calcul de la déformation d'un objet selon des lois physiques, formulé initialement par Terzopoulos *et al.* [1], consiste à déterminer le mouvement de cet objet d'après les forces internes et externes qui lui sont appliquées. Dans un contexte d'animation, comme celui d'une simulation en réalité virtuelle, il s'agit de déterminer la position de chaque point de l'objet, à un certain moment, à partir de l'état de la simulation au moment précédent. Le passage de l'état de la simulation d'un moment à un autre est appelé le *rafraîchissement*, parfois *mise à jour*, de la simulation et l'intervalle entre les deux moments le *pas de temps*.

La position de l'objet dans le temps est donnée par $\boldsymbol{x}$ dans l'équation 1.1. Cependant, pour la résoudre numériquement, on discrétise l'objet en un ensemble de nœuds sur lesquels on trouve une solution approximative de cette équation. On obtient alors un système d'équations :

$$\boldsymbol{M}\ddot{\boldsymbol{u}}(t) - \boldsymbol{K}\boldsymbol{u}(t) = \boldsymbol{f}^{ext}(t), \tag{2.1}$$

où $\boldsymbol{u}$ représente le vecteur des déplacements des nœuds, avec $\boldsymbol{u}_i(t) = \boldsymbol{x}_i(t) - \boldsymbol{x}_i(0)$, $\boldsymbol{M}$ la distribution de la masse de l'objet parmi les nœuds, $\boldsymbol{K}$ la matrice de rigidité du système et $\boldsymbol{f}^{ext}$ le vecteur des forces nodales. $\ddot{\boldsymbol{u}}$ représente la double dérivée temporelle de $\boldsymbol{u}$. Il est important de noter que dans le cas général, la matrice $\boldsymbol{K}$ dépend de la déformation de l'objet, telle que déterminée par $\boldsymbol{u}$. Le terme $\boldsymbol{K}\boldsymbol{u}$ représente les forces élastiques nodales générées par la déformation de l'objet.

Les différentes méthodes présentées dans cette section sont toutes des façons différentes de résoudre l'équation 2.1, en particulier pour l'intégration de $\ddot{\boldsymbol{u}}$ dans le temps et le calcul des forces élastiques $\boldsymbol{K}\boldsymbol{u}$. Nous définissons ici quelques termes et leur notation afin de faciliter la discussion.

La *déformation* d'un objet est une mesure des changements dans l'espacement relatif entre les points de l'objet. Elle est représentée par un tenseur $\boldsymbol{\varepsilon}(\boldsymbol{u})$ qui dépend du déplacement des points, plus spécifiquement du gradient $\nabla\boldsymbol{u}$ de ce déplacement.

Les *contraintes* sont les forces élastiques issues de la déformation de l'objet. Elles sont représentées par un tenseur $\boldsymbol{\sigma}(\boldsymbol{\varepsilon})$ qui dépend de la déformation. La relation entre les contraintes et la déformation est déterminée par la *loi de comportement*.

Pour effectuer les calculs, les nœuds sont regroupés par une *connectivité*, qui détermine leur

relation et certaines de leurs propriétés. En particulier, une *fonction de forme* ϕ_i, dépendant de la connectivité, est associée à chaque nœud i. Les fonctions de forme permettent d'estimer la valeur d'un champ quelconque en n'importe quel point de l'objet, tant que cette valeur est connue sur les nœuds. Par exemple, pour connaître la valeur du déplacement en un point arbitraire, on utilise

$$\boldsymbol{u}^h(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{u}_i \phi_i(\boldsymbol{x}), \tag{2.2}$$

où $\boldsymbol{u}^h(\boldsymbol{x})$ est l'estimation de la position au point $\boldsymbol{x}$, $\boldsymbol{u}_i$ la valeur du déplacement au nœud i et $\phi_i(\boldsymbol{x})$ la valeur de la fonction de forme du nœud i au point $\boldsymbol{x}$.

Les fonctions de formes nous donnent également le moyen de calculer le produit $\boldsymbol{K}\boldsymbol{u}$. Les termes de la matrice $\boldsymbol{K}$ sont donnés par

$$\boldsymbol{K}_{ij} = \int_{\Omega} \boldsymbol{B}_i^T \boldsymbol{C} \boldsymbol{B}_j d\Omega, \tag{2.3}$$

où

$$\boldsymbol{B}_i = \begin{Bmatrix} \frac{\partial \phi_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial \phi_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial \phi_i}{\partial z} \\ \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial x} & 0 \\ 0 & \frac{\partial \phi_i}{\partial z} & \frac{\partial \phi_i}{\partial y} \\ \frac{\partial \phi_i}{\partial z} & 0 & \frac{\partial \phi_i}{\partial x} \end{Bmatrix} \tag{2.4}$$

et $\boldsymbol{C}$ est une matrice qui décrit la loi de comportement.

Plusieurs approches existent pour réduire le temps de calcul. La méthode lagrangienne totale [2] évalue l'intégrale de l'équation 2.3 par rapport au volume initial – non déformé – de l'objet, plutôt que son volume courant. Il n'est donc pas nécessaire de calculer un nouveau volume à chaque pas de temps, ce qui réduit de façon importante la quantité de calcul durant la simulation en conservant en mémoire le volume calculé à l'initialisation.

Un autre moyen fréquemment employé pour garder la quantité de calcul au minimum consiste à utiliser une description linéaire en $\boldsymbol{u}$ de la déformation $\boldsymbol{\varepsilon}(\boldsymbol{u})$, appelée petit tenseur [3], ainsi qu'une loi de comportement linéaire. La matrice $\boldsymbol{K}$ devient alors constante plutôt que de dépendre sur $\boldsymbol{u}$ et n'a donc pas besoin d'être calculée à chaque pas de temps.

Par contre, une description linéaire de la déformation introduit des forces *fantômes* lors de grandes déformations ou de rotations dans l'objet simulé, qui causent une expansion de son

volume [3]. Pour éviter cet effet tout en conservant une formulation linéaire de $\boldsymbol{\varepsilon}$, il est possible d'employer une méthode de *corotation* [4]. La composante rotative de la déformation d'un élément est retirée avant de calculer les forces sur celui-ci, puis elle est appliquée sur les forces avant de calculer le déplacement nodal. Ainsi, plutôt que d'évaluer entièrement la matrice $\boldsymbol{K}$ à chaque pas de temps, il suffit d'extraire la rotation de chaque élément pour l'appliquer aux termes de l'expression 2.4.

Une fois le terme $\boldsymbol{Ku}$ de l'équation 2.1 calculé, il est possible d'isoler le terme $\ddot{\boldsymbol{u}}$ et d'en calculer l'intégrale dans le temps. Une méthode explicite nous permet de déterminer directement $\boldsymbol{u}$ à partir de $\ddot{\boldsymbol{u}}$, avec par exemple

$$\begin{aligned} \dot{\boldsymbol{u}}(t) = \dot{\boldsymbol{u}}(t) &= \frac{\boldsymbol{u}(t+\Delta t) - \boldsymbol{u}(t)}{\Delta t} \\ \boldsymbol{a}(t) = \dot{\dot{\boldsymbol{u}}}(t) &= \frac{\dot{\boldsymbol{u}}(t+\Delta t) - \dot{\boldsymbol{u}}(t)}{\Delta t} \end{aligned} \tag{2.5}$$

Dans l'équation 2.5, la dérivée dans le temps du déplacement et de la vitesse est estimée par les différences finies.

En contraste, une méthode implicite donne lieu à un système non linéaire :

$$\begin{aligned} \dot{\boldsymbol{u}}(t+\Delta t) = \dot{\boldsymbol{u}}(t+\Delta t) &= \frac{\boldsymbol{u}(t+\Delta t) - \boldsymbol{u}(t)}{\Delta t} = \frac{\Delta \boldsymbol{u}}{\Delta t} \\ \boldsymbol{a}(t+\Delta t) = \dot{\dot{\boldsymbol{u}}}(t+\Delta t) &= \frac{\dot{\boldsymbol{u}}(t+\Delta t) - \dot{\boldsymbol{u}}(t)}{\Delta t} = \frac{\Delta \dot{\boldsymbol{u}}}{\Delta t} \end{aligned} \tag{2.6}$$

Cette méthode est beaucoup plus longue à résoudre étant donné que la valeur de la position à un temps t dépend de la vitesse à ce même temps. Cependant, contrairement à la méthode explicite, elle donne lieu à une solution inconditionnellement stable [5].

Il est possible de contourner la difficulté de l'intégration dans le temps en résolvant un système statique

$$\boldsymbol{Ku} = \boldsymbol{f}^{ext}. \tag{2.7}$$

Cette façon de faire peut donner lieu à une gestion complètement différente des interactions avec l'utilisateur. En effet, plutôt que de chercher à connaître la nouvelle position de l'objet simulé déformé, on impose plutôt un ensemble de déplacements nodaux déterminés par la position de l'outil, et la déformation résultante donne les forces de contact sur les nœuds déplacés.

Les sections suivantes présentent une revue des différentes méthodes de déformation utilisées pour simuler des objets, en particulier dans le cadre de simulations chirurgicales incluant la capacité de découper. Les différents solveurs utilisés pour approximer une solution à l'équa-

tion 1.1 seront discutés dans la section 2.2, après que les différentes approches aient été décrites. En particulier, des travaux qui implantent les solveurs sur GPU seront présentés.

2.1.1 Éléments finis avec remaillage

La méthode des éléments finis discrétise le domaine de calcul en des formes géométriques simples, dont les propriétés sont bien connues. Cette approche facilite beaucoup l'intégration sur le volume de chaque élément, en particulier avec les éléments réguliers comme des tétraèdres ou hexaèdres. Les difficultés surviennent lorsque l'objet modélisé ainsi est découpé : les éléments ne sont plus valides, alors il faut les redéfinir. Cette redéfinition des éléments dans la région découpée nécessite certaines précautions, étant donné que des éléments mal conditionnés – trop minces, par exemple – vont causer des instabilités et une perte de précision [6, 7].

Plusieurs méthodes existent pour effectuer cette modification des éléments. Les plus simples ne modifient pas le nombre d'éléments ; ils sont simplement séparés le long de leurs faces en dupliquant les nœuds qu'ils ont en commun, comme Müller et Gross [8] ou Yeung *et al.* [9]. Le problème le plus évident avec cette technique est que la séparation ne suit pas précisément la ligne de coupe effectuée par l'utilisateur. En ajustant la position des nœuds, une technique souvent appelée *node snapping*, on obtient une coupure beaucoup plus nette. Cette méthode est adoptée dans plusieurs simulateurs [10–13]. Cependant, la possibilité d'éléments mal formés demeure.

Une autre approche simple consiste à retirer des éléments du maillage [14–18]. Les deux inconvénients principaux de cette méthode sont que la séparation ne suit pas fidèlement la coupe par l'utilisateur et que l'objet découpé perd du volume à chaque modification. Nakao *et al.* [19] évite la perte de matière en ajustant les éléments voisins de celui retiré le long de la ligne de coupe. Pour réduire l'effet de la perte de volume, il est également possible de d'abord subdiviser de façon régulière l'élément à modifier, puis retirer seulement des éléments de plus petite taille [20]. Une autre option consiste à retirer progressivement de la matière d'un élément et en ajuster les propriétés, sans enlever cet élément au complet, comme Jeřábková *et al.* [21] ou Delorme *et al.* [22].

La manière la plus réaliste et précise de modifier le maillage d'éléments finis demeure toutefois la subdivision de ces éléments précisément le long de la coupure. Lorsqu'ils sont tétraédriques, le découpage requiert une augmentation importante du nombre d'éléments à chaque fois. Pour un découpage générique, un seul tétraèdre en devient 17 [23]. En tenant compte de la topologie spécifique de la coupure sur l'élément, Mor et Kanade [24] crée seulement 5 à 9 nouveaux éléments. C'est l'approche adoptée entre autres par Courtecuisse *et al.* [25, 26] et

Li *et al.* [27]. Nienhuys [28] combine cette approche avec le *node snapping* et étudie en détail l'effet des changements sur la qualité de la simulation.

Pour éviter une croissance trop importante du nombre d'éléments, des éléments polyédriques ont été développés [29, 30]. Cette addition permet de simplement diviser un élément – tétraédrique ou non – en deux, peu importe la ligne de coupe. Cependant, elle n'évite pas les problèmes d'éléments mal conditionnés. Paulus *et al.* [31] élabore une méthode de subdivision qui crée uniquement des tétraèdres et évite complètement les éléments mal conditionnés en considérant une approximation de la coupure. Cakir *et al.* [32] et Zhang *et al.* [33] utilisent par ailleurs un mélange de découpage et d'ajustement de nœud pour garder l'augmentation du nombre d'éléments au minimum tout en réduisant la possibilité d'éléments mal conditionnés.

Pour éliminer complètement le problème de conditionnement des éléments lors de changements de topologie, Molino *et al.* [6] implante un algorithme de *nœud virtuel*. Cette méthode consiste à dupliquer l'élément à découper, puis assigner à chaque copie les propriétés correspondant à la quantité de matière qui l'occupe. Elle a été améliorée par Sifakis *et al.* [34] en permettant que le fragment d'un élément puisse ne contenir aucun des nœuds initiaux de cet élément. Ainsi, un objet peut être subdivisé indéfiniment, sans le moindre problème relié au conditionnement des éléments du maillage. Cette approche introduit une distinction entre le maillage d'éléments finis et la géométrie du modèle simulé qui est primordiale pour le découpage avec la méthode des éléments finis composites, discutée ci-dessous.

2.1.2 Approches multirésolutions

Pour optimiser le temps de calcul – dans le but d'avoir du temps réel – on peut utiliser des méthodes dites hybrides, dans lesquelles le modèle est subdivisé en une région opérable et une non opérable. La région non opérable, importante pour la crédibilité de la simulation, mais dont la précision du comportement n'est pas essentielle, est simulée avec un modèle physique simplifié ou des éléments plus grossiers. En contraste, la région opérable, qui nécessite une plus grande précision pour les manipulations chirurgicales, permet une interaction plus précise et réaliste, avec la possibilité d'être découpée [14, 35].

Le concept d'utiliser plusieurs résolutions d'éléments est poussé plus loin par Debunne *et al.* [36], qui présente une méthode ou le modèle entier est discrétisé à plusieurs résolutions et où la déformation est calculée à la résolution nécessaire pour obtenir une certaine précision, tout en restant dans un certain budget de calcul. Cette méthode ne permet cependant pas de découpage.

Miyazaki *et al.* [37] et Jeřábková *et al.* [21] créent un modèle déformable multirésolution

à topologie modifiable en regroupant des éléments de très petite taille en des formes plus grossières et en superposant ces deux résolutions. Lorsque vient le temps de découper, les éléments grossiers sont décomposés en leurs plus petites parties afin d'obtenir une coupure très précise. Miyazaki *et al.* [37] sépare simplement les éléments les plus fins le long de leurs faces. Jeřábková *et al.* [21] utilise plutôt une méthode semblable à celle du nœud virtuel : initialement, les éléments les plus fins traversés par l'outil sont simplement retirés, mais lorsqu'un élément grossier devient divisé en plusieurs parties séparées, il est dupliqué et la matière est répartie entre les copies, selon la configuration de la coupure déterminée par les éléments fins.

L'approche de réduction du nombre de degrés de liberté en regroupant des éléments fins pour former des éléments de plus en plus grossiers est formalisée par le concept d'éléments finis composites (CFE) [38]. Les éléments composites ont des propriétés différentes selon les éléments dont ils ont été formés et peuvent être partiellement remplis, comme dans Jeřábková *et al.* [21]. Cela permet de préserver le comportement de l'objet global avec les propriétés des éléments fins. Cette méthode est reprise avec succès par plusieurs auteurs, en imbriquant des éléments tétraédriques dans une grille hexaédrique [39] ou simplement des éléments cubiques dans une grille hexaédrique [40–43]. Wu *et al.* [44] permet par ailleurs de détecter des contacts à une résolution plus fine que les plus petits éléments en décrivant ceux-ci à l'aide d'un champ de distance.

2.1.3 Éléments finis étendus (XFEM)

Pour modéliser les discontinuités dans le champ de déplacement sans affecter le maillage, il est possible *d'enrichir* un élément en lui ajoutant des degrés de liberté dont la fonction de forme est discontinue [45]. La discrétisation du champ de déplacement de l'équation 2.2 devient alors

$$\boldsymbol{u}^h(x) = \sum_{i=1}^{N} \phi_i(x) \left(u_i + \sum_{j=1}^{n^{E_i}} g_j(x) a_{ji} \right). \tag{2.8}$$

Cette méthode a été utilisée avec succès pour simuler la propagation de fissure dans un objet [46–48] et dans le cadre d'une simulation chirurgicale [49]. Quesada *et al.* [50] combine un découpage par XFEM à un modèle réduit de corps déformable. L'inconvénient principal de la méthode XFEM réside dans la difficulté de décrire des coupures multiples dans le même élément, lorsque les coupures s'entrecroisent.

2.1.4 Éléments finis de frontière

Les méthodes présentées jusqu'à présent effectuent toutes le calcul de déformation et des forces élastiques sur le volume de l'objet. Elles résolvent directement la forme variationnelle de l'équation 1.1. Cependant, l'intégrale volumique de cette forme peut être convertie en intégrale surfacique à l'aide du théorème de Green [51]. En partant de la forme surfacique de l'équation, seule la surface de l'objet à simuler doit être discrétisée, ce qui réduit de façon importante le nombre de nœuds nécessaires pour décrire complètement l'objet. Par contre, le maillage doit toujours être recalculé lors de changements de topologie. Cette méthode est utilisée notamment par Wang *et al.* [52] et Choi *et al.* [53].

2.1.5 Particules

Les méthodes présentées ci-dessus ont toutes en commun le défi que représentent les changements de topologie étant donné qu'elles dépendent d'un maillage sur lequel les calculs sont structurés. Lorsque la topologie de l'organe est modifiée, il est nécessaire d'ajuster le maillage en créant de nouveaux éléments, ce qui cause un changement dans le système matriciel à résoudre (eq. 2.1). Les changements incluent souvent un ajout de nœuds, ce qui résulte en une augmentation de la taille des matrices et vecteurs du système. Plutôt que d'assembler les nœuds sous forme d'éléments, il est possible de les considérer comme un nuage de *particules*, connectées de façon plus flexible, et ainsi d'éliminer partiellement ou complètement la dépendance du système d'équations sur un maillage.

La principale différence avec les méthodes par éléments est dans l'approximation du champ de déplacement $\boldsymbol{u}^h(\boldsymbol{x})$. Avec des éléments – par exemple des tétraèdres – le support de la fonction de forme ϕ_i dans l'équation 2.2 s'étend uniquement sur les éléments dont fait partie le nœud i. Avec des particules, la fonction de forme est plutôt définie sur un certain rayon de support h_i, et le déplacement de chaque point à l'intérieur de ce rayon est influencé par le nœud i.

Fonctions de forme

Desbrun et Gascuel [54] applique le concept de la méthode *Smoothed Particle Hydrodynamics* (SPH) pour représenter les déformations. Initialement utilisée pour modéliser des problèmes d'astrophysique, cette méthode représente les propriétés d'un matériau par des points dont la valeur est *lissée* dans un certain rayon [55]. Chaque particule possède des propriétés fixes, comme une masse, une position et une vélocité, qui sont étalées dans l'espace selon une fonction noyau w, dont la valeur tombe à zéro au-delà d'un rayon h et dont l'intégrale sur

la sphère de rayon h est de 1. Par exemple, l'approximation du champ de déplacement $\boldsymbol{u}$ devient

$$\boldsymbol{u}^h(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{u}_i w(\boldsymbol{x} - \boldsymbol{x}_i), \tag{2.9}$$

où le noyau $w(\boldsymbol{x} - \boldsymbol{x}_i) \equiv w_i(\boldsymbol{x})$ est directement la fonction de forme ϕ_i et où N est le nombre de particules dans un rayon h autour du point $\boldsymbol{x}$. Le gradient du déplacement $\nabla \boldsymbol{u}$ est obtenu à travers le gradient du noyau :

$$\nabla \boldsymbol{u}^h(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{u}_i \nabla w_i(\boldsymbol{x}). \tag{2.10}$$

Une autre méthode populaire pour calculer le champ de déformation consiste à utiliser une approximation par les moindres carrés (MLS) [56]. Le déplacement à un point $\boldsymbol{x}$ est représenté par un polynôme dont les coefficients sont calculés lors de l'initialisation de la simulation :

$$\boldsymbol{u}^h(\boldsymbol{x}) = \boldsymbol{p}^T(\boldsymbol{x})\boldsymbol{a}(\boldsymbol{x}), \tag{2.11}$$

où $\boldsymbol{p}$ est la base du polynôme, par exemple $[1\ x\ y\ z]$ pour une interpolation linéaire, et $\boldsymbol{a}$ constitue le vecteur de coefficients de ce polynôme. Les coefficients $\boldsymbol{a}(\boldsymbol{x})$ sont donnés par

$$\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{A}^{-1}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}, \tag{2.12}$$

où

$$\boldsymbol{A}(\boldsymbol{x}) = \sum_{i=1}^{N} w_i(\boldsymbol{x})\boldsymbol{p}^T(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i), \tag{2.13}$$

et $\boldsymbol{p}(\boldsymbol{x})$ est le vecteur des $w_i(\boldsymbol{x})\boldsymbol{p}(\boldsymbol{x}_i)$, avec chaque i correspondant à une particule dans le voisinage du point $\boldsymbol{x}$. Les détails des calculs menant à cette approximation sont exposés dans l'annexe A.

En ce qui concerne la précision, l'approximation MLS linéaire a une consistance de premier ordre, ce qui signifie qu'elle peut représenter exactement un champ de déplacement qui varie linéairement, contrairement à la méthode SPH qui n'a une consistance que de degré 0 [55]. Il est possible d'obtenir une meilleure approximation en augmentant le degré de la base polynomiale $\boldsymbol{p}$. Le désavantage de la méthode MLS est que le calcul des coefficients MLS requiert l'inversion de la matrice de moment $\boldsymbol{A}$ (équation 2.13), ce qui est impossible lorsque le voisinage d'un point ne contient pas assez de nœuds, ou lorsque ces nœuds sont colinéaires ou coplanaires [57].

D'autres méthodes ont été développées pour construire les fonctions d'interpolation, généra-

lement à partir de bases radiales (comme SPH) ou polynomiales (comme MLS). Elles sont cependant plus rarement utilisées pour résoudre des problèmes d'élasticité en temps réel. La méthode d'interpolation ponctuelle (PIM) [58] en particulier a été implantée dans un contexte de simulation chirurgicale. Elle sera décrite plus loin dans ce document étant donné qu'elle ne dépend pas uniquement des particules, mais aussi d'un maillage d'intégration superposé sur le domaine. Une autre approche est implantée par Martin *et al.* [59], qui utilise une généralisation de la méthode MLS pour inclure les dérivées le long des axes dans la base polynomiale. Les fonctions de formes sont alors toujours calculables, peu importe le voisinage d'un point. Par contre, le temps de calcul associé à cette méthode la rend peu pratique pour une application en temps réel.

Imposition des conditions aux frontières

L'imposition de conditions aux frontières est souvent problématique avec les méthodes particulaires étant donné qu'elle ne satisfont généralement pas la propriété du delta de Kronecker – à l'exception des fonctions d'interpolation PIM présentées à la section 2.1.7.

Les conditions essentielles peuvent être appliquées à l'aide de multiplicateurs de Lagrange [56]. Le système à résoudre devient alors

$$\begin{bmatrix} \boldsymbol{K} & \bar{\boldsymbol{K}} \\ \bar{\boldsymbol{K}}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \bar{\boldsymbol{u}} \end{bmatrix}, \tag{2.14}$$

où $\bar{\boldsymbol{K}}$ contient 1 dans les colonnes correspondant à une particule où une position est imposée et 0 ailleurs, $\bar{\boldsymbol{u}}$ correspond aux déplacements imposés et $\boldsymbol{\lambda}$ contient les forces correspondant à ces déplacements une fois le système résolu.

2.1.6 Méthode basée sur les nœuds (PBA)

La façon la plus simple de déterminer la force élastique en un point consiste à prendre la somme des forces appliquées par chaque particule voisine sur ce point. Desbrun et Gascuel [54] calcule une force d'attraction-répulsion élastique semblable aux potentiels de Lennard-Jones [55], dont la valeur est dérivée de l'équation des gaz parfaits. Cette force élastique pousse un objet à retrouver son volume initial lorsqu'il est déformé, à une vitesse proportionnelle à une constante k, qui détermine la rigidité de l'objet. Les auteurs ne précisent cependant pas comment cette constante est reliée aux propriétés élastiques d'objets physiques.

Müller *et al.* [57] dérive une formulation particulaire des forces sur chaque nœud basée sur la mécanique des solides plutôt que des fluides, de la même façon qu'avec les éléments finis.

Le modèle de déformation obtenu est donc plus réaliste que Desbrun et Gascuel [54]. Les auteurs appliquent une approximation MLS directement sur le gradient $\nabla \boldsymbol{u}$ de la déformation à partir d'une approximation par polynôme de Taylor du déplacement $\boldsymbol{u}$ à un nœud $\boldsymbol{x}_j$ dans le voisinage de $\boldsymbol{x}_i$, pour des valeurs connues de $\boldsymbol{u}_i$ et $\boldsymbol{u}_j$:

$$\tilde{\boldsymbol{u}}(\boldsymbol{x}_j) = \boldsymbol{u}_i + \nabla \boldsymbol{u}\Big|_{\boldsymbol{x}_i} \cdot (\boldsymbol{x}_j - \boldsymbol{x}_i). \tag{2.15}$$

En choisissant l'erreur d'approximation e comme la somme pondérée des différences du champ de déplacement tel que

$$e = \sum_j^N (\tilde{\boldsymbol{u}}_j - \boldsymbol{u}_j)^2 w_j(\boldsymbol{x}_i) \tag{2.16}$$

et en minimisant cette erreur, on obtient un système linéaire

$$\nabla \boldsymbol{u}(\boldsymbol{x}_i) = \boldsymbol{A}^{-1} \left(\sum_{j=1}^N (\boldsymbol{u}_j - \boldsymbol{u}_i)(\boldsymbol{x}_j - \boldsymbol{x}_i) w_j(\boldsymbol{x}_i) \right), \tag{2.17}$$

où

$$\boldsymbol{A} = \sum_{j=1}^N (\boldsymbol{x}_j - \boldsymbol{x}_i)(\boldsymbol{x}_j - \boldsymbol{x}_i)^T w_j(\boldsymbol{x}_i). \tag{2.18}$$

Une fois le gradient du déplacement déterminé à un point, sa déformation, puis ses contraintes et la force qu'il applique à ses voisins peuvent être calculées.

Keiser *et al.* [60] combine le modèle de Müller *et al.* [57] avec des particules liquides simulées par SPH. La simulation résultante permet de passer d'un état – solide ou liquide – à un autre en modulant simplement la contribution des forces élastiques, de viscosité, de pression et de tension agissant sur une particule. Ainsi, bien que ces travaux ne traitent pas directement du découpage des objets simulés, ils permettent certains changements de topologie.

Solenthaler *et al.* [61] et Schläfli [62] implantent le modèle de déformation PBA avec des fonctions de forme SPH pour éviter les problèmes de petit voisinage ou de points coplanaires reliés à l'approximation MLS, étant donné que ceux-ci se produisent fréquemment lors de changements de topologie. De plus, plutôt que de considérer globalement la configuration non déformée, les auteurs choisissent une mesure locale de la forme initiale de l'objet, ce qui facilite le traitement de la déformation lorsqu'un objet est divisé en plusieurs parties complètement séparées.

Gerszewski *et al.* [63] réduit les problèmes de l'approximation MLS en calculant la transformation par moindres carrés à chaque pas de temps et en composant les transformations à chaque fois. Cela permet de traiter correctement les changements de voisinage des nœuds

lors des modifications à la topologie, mais au coût d'une dérive progressive du gradient de déformation avec le temps.

Becker *et al.* [64] présente une méthode corotative similaire à celle utilisée avec les éléments finis. La rotation en un point est extraite selon une approche SPH, c'est-à-dire en additionnant les rotations des particules individuelles du voisinage du point. Les rotations individuelles sont obtenues d'après le voisinage initial – non déformé – de chaque nœud. Cette méthode permet d'utiliser un tenseur de déformation linéaire et d'obtenir plus facilement une performance en temps réel.

Lorsque des modèles sans maillage sont utilisés, la topologie de l'objet est initialement décrite uniquement par la proximité des particules. Afin de pouvoir simuler des changements tels que ceux introduits par la coupure d'un scalpel, un critère additionnel, autre que la simple distance euclidienne, doit être utilisé pour déterminer quelles particules sont dans le voisinage, ou le rayon d'influence, de chacune. Sous sa forme la plus simple, on parle d'un critère de *visibilité* [65]. Lorsqu'un obstacle – une discontinuité – se trouve entre deux particules proches, on considère que les particules ne sont pas mutuellement visibles et ne s'influencent donc pas.

Le critère de visibilité peut toutefois donner lieu à des discontinuités additionnelles [66]. Pour éviter des discontinuités trop brusques, il est possible de définir la distance entre deux particules par la plus courte ligne ininterrompue qui relie ces particules. Ce critère dit de *diffraction* peut être approximé de façon efficace avec un graphe de visibilité [67]. D'autres critères ont été élaborés pour raffiner la description des discontinuités, comme la transparence ou le critère de visibilité étendu [66].

Peng *et al.* [68] ajoute une surface explicite sur le modèle PBA et utilise cette surface pour déterminer si chaque paire de points proches s'influence ou pas, en utilisant le critère de visibilité simple. Cette méthode lui permet de simuler interactivement la déformation d'un œil lors d'incisions dans la cornée.

Steinemann *et al.* [67] développe une méthode pour simuler des craques et des coupures sur un modèle PBA. Un graphe de visibilité est construit à l'initialisation pour déterminer quels nœuds sont dans le voisinage de chacun. Lors de changements de topologie, la coupure est d'abord modélisée explicitement, puis le graphe de visibilité est mis à jour selon les intersections entre ses arêtes et la coupure. Deux nœuds qui ne sont plus directement connectés peuvent toujours contribuer à leurs fonctions de forme respectives s'ils sont tous les deux reliés à un troisième nœud et que la longueur totale de ces liens est inférieure au rayon d'influence des particules. Ce critère qui considère la distance entre deux particules comme la longueur du chemin le plus court qui les relie est appelé critère de diffraction. Les auteurs ont utilisé cette méthode pour simuler interactivement la résection de polypes dans l'utérus.

Plus récemment, cette méthode a été utilisée pour simuler une lobectomie hépatique et la résection d'une tumeur cérébrale [69]

Pietroni *et al.* [66] propose un critère de visibilité étendu, avec lequel le lien entre deux nœuds est constitué d'une paire de cônes plutôt que d'une simple ligne. Une coupure réduit progressivement le cône de visibilité, ce qui permet de quantifier la visibilité plutôt que d'utiliser une valeur binaire. Ce choix est également motivé par la facilité relative de son implantation sur GPU. En effet, l'intersection des deux cônes définit un disque qui peut être modélisé par une texture gérée complètement par la carte graphique. L'efficacité de cette méthode est démontrée lors d'une simulation de découpage interactif d'un foie, à une résolution environ deux fois plus élevée que Steinemann *et al.* [67].

Plus récemment, Aras *et al.* [70] adopte une approche similaire aux XFEM pour découper un modèle PBA. Le champ de déplacement est discrétisé par l'équation 2.8, avec les fonctions de forme ϕ_i calculées par MLS. Les fonctions d'enrichissement sont plutôt calculées sur une grille d'enrichissement, où chaque segment de coupe est assigné à un point et où une fonction de distance est évaluée en coordonnées locales à la coupure. L'utilisation d'une grille permet d'éviter les instabilités en présence de plusieurs coupures séparées et d'accélérer l'identification des nœuds affectés par une coupure. La méthode est démontrée en simulant la coupe d'un bloc rectangulaire déformable à un taux élevé (plus de 100 Hz).

2.1.7 Particules avec grille d'intégration

Les méthodes particulaires présentées ci-dessus sont considérées purement sans maillage : l'intégration des forces (équation 2.3) se fait directement sur chaque nœud, puis la force calculée est distribuée aux voisins des nœuds. La principale faiblesse de cette démarche quant à la précision se trouve dans le calcul du volume de chaque particule, qui est le domaine de l'intégrale. En effet, ce volume doit être calculé à partir de la densité de la particule, qui est elle-même estimée d'après les masses des particules voisines pondérées par une fonction noyau, selon l'approche SPH. La masse elle-même est estimée d'après la distance avec les particules voisines, pondérées par un facteur choisi pour que la densité finale soit proche de la densité du matériau simulé [57].

La méthode de Galerkin sans éléments (EFG), introduite par Belytschko *et al.* [56] pour les problèmes d'élasticité, fait appel à une grille d'intégration indépendante des particules. Les éléments de cette grille ont un volume bien défini et permettent une intégration par quadrature de Gauss. Le nombre de points d'intégration, ou points de Gauss, dans un élément est déterminé par le nombre de nœuds qui se trouvent dans cet élément. La déformation et les contraintes sont calculées à chaque point d'intégration d'après le déplacement des nœuds

voisins avec une interpolation par MLS. Les contraintes servent à calculer une force *locale* sur les nœuds voisins, qui accumulent les forces calculées à chaque point d'intégration dont ils sont dans le voisinage [71].

En termes de performance, cette méthode peut obtenir des résultats très similaires à une méthode purement sans maillage. Horton *et al.* [71] montre que le nombre de points d'intégration nécessaires dépend de l'uniformité de la distribution des particules. Dans le meilleur des cas, un rapport d'un seul point d'intégration par nœud suffit.

Brunet *et al.* [72] combine la corotation avec la méthode EFG pour permettre l'utilisation d'un modèle linéaire de déformation. Dans ces travaux, les particules sont également structurées le long d'une grille régulière et une estimation du volume des éléments d'intégration représente plus fidèlement la géométrie de l'objet modélisé, permettant alors que des particules se retrouvent à l'extérieur de la géométrie.

Jung et Lee [73] implante la méthode EFG dans un contexte de simulation chirurgicale. Les auteurs développent une structure de graphe pour décrire les relations topologiques entre les particules qui permet la simulation de découpage selon un critère de visibilité. Lorsqu'une particule n'est plus visible par une autre, elle n'entre plus dans le calcul de sa déformation. Les seuls changements nécessaires aux données de la simulation consistent à mettre à jour le graphe de visibilité, c'est-à-dire la liste des voisins des nœuds découpés, et les fonctions de forme qui leur sont associées.

Jin *et al.* [74] ajoute la possibilité de découper l'objet simulé à la méthode présentée par Horton *et al.* [71]. Plutôt que d'utiliser un graphe pour déterminer l'influence entre les nœuds, les auteurs définissent une fonction de surface de niveau. Cette fonction modélise la coupure en définissant la distance entre un point et la surface de la coupure, ce qui permet de déterminer directement si deux points sont séparés. Cheng *et al.* [75]

Dehghan *et al.* [76] compare les résultats obtenus par une implantation de la méthode EFG avec les déformations réelles sur un cube de silicone. Les auteurs montrent que la méthode entraîne une erreur semblable à celle obtenue avec des éléments finis pour un même nombre de degrés de liberté. Ils appliquent la méthode pour simuler interactivement la déformation sans coupe d'un foie et d'une vésicule biliaire.

Liu *et al.* [58] décrit une méthode d'interpolation ponctuelle similaire à EFG, mais avec des

fonctions de forme constituées à la fois de composantes radiale et polynomiale, telles que

$$\begin{aligned} \boldsymbol{u}^h(\boldsymbol{x}) &= \sum_{i=1}^{N} R_i(\boldsymbol{x}) a_i + \sum_{j=1}^{M} p_j(\boldsymbol{x}) b_j \\ &= \boldsymbol{R}^T(\boldsymbol{x})\boldsymbol{a} + \boldsymbol{p}^T\boldsymbol{b} \\ &= \boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{U}_e \\ &= \sum_{i=1}^{N} \phi_i(\boldsymbol{x}) u_i, \end{aligned} \tag{2.19}$$

où N correspond au nombre de nœuds dans le voisinage du point $\boldsymbol{x}$, chacun ayant pour fonction de forme une fonction radiale R_i, et M à la dimension de la base polynomiale $\boldsymbol{p}$ utilisée. La partie polynomiale est optionnelle ; dans le cas où elle est omise, l'approximation devient identique à celle de la méthode SPH.

Cette façon de déterminer les fonctions de forme comporte plusieurs avantages par rapport à la méthode SPH ou MLS. Les fonctions radiales sont stables et fonctionnent avec des distributions irrégulières de particules, tandis que l'ajout de termes polynomiaux pour leur calcul permet d'obtenir une consistance du degré désiré. De plus, les fonctions résultantes possèdent la propriété du delta de Kronecker, ce qui permet d'appliquer facilement les conditions essentielles aux limites.

Zou *et al.* [77] implante la méthode PIM pour un simulateur de chirurgie en temps réel. Tout comme Liu *et al.* [58], les auteurs utilisent une grille d'intégration pour calculer la force sur les nœuds. Cependant, plutôt que l'approche statique, ils utilisent l'équation dynamique d'élasticité, à laquelle ils ajoutent une composante viscoélastique. Leur modèle permet de prédire de façon très précise la force et la déformation résultant de la compression de tissu de foie de porc.

2.1.8 Collocation

Un désavantage important des méthodes particulaires par rapport aux éléments finis est la quantité de calculs nécessaire pour obtenir un même niveau de précision. Dans un contexte de simulation interactive, le critère de temps réel est essentiel pour permettre une utilisation fluide du simulateur, et il est plus difficile de l'atteindre sans maillage.

Les méthodes de collocation évitent l'intégration coûteuse de l'équation 2.4 en discrétisant directement la formulation forte du problème de déformation. L'approximation de la solution est donnée par la même équation que 2.2, mais avec des fonctions d'interpolation ϕ_i à support global [78]. La solution estimée résout l'équation aux points de collocation et est interpolée

sur le reste du domaine. Les fonctions d'interpolation sont généralement des fonctions radiales (RBF).

De *et al.* [79] applique une méthode de collocation pour une simulation chirurgicale en temps réel en utilisant des fonctions de base à support local appelées *sphères finies*. L'approximation est calculée à l'aide des MLS. La déformation est simulée seulement localement, dans une zone autour de la pointe de l'outil lorsqu'il entre en contact avec l'organe. Les nœuds de sphères finies sont distribués dans le voisinage de l'outil pour donner lieu à une discrétisation locale du volume de l'organe. Des déplacements sont prescrits dans le voisinage immédiat du point de contact et le système matriciel est résolu pour obtenir le reste des déplacements. Le problème résolu est uniquement la version statique de l'équation 1.1 ; il est décrit plus en détail dans la section 2.4.

De et Lim [80] exploite l'idée que les déformations perceptibles par l'utilisateur se trouvent toutes dans un certain rayon autour de l'outil. Les nœuds de calculs sont regroupés dans un nuage qui se déplace avec l'outil. La déformation est calculée uniquement à l'intérieur de la zone d'influence de l'outil. Les auteurs incorporent également des propriétés non linéaires de matériau en simulant un modèle linéaire globalement et en lui ajoutant une composante non linéaire uniquement dans la zone d'influence de l'outil.

De et Lim [80] et Banihani *et al.* [81] augmentent la performance de la méthode de collocation en utilisant une approche de réduction de modèle, qui diminue le nombre de degrés de liberté à calculer à chaque pas de temps. Cette approche permet de simuler globalement en temps réel des modèles au comportement non linéaire, sans toutefois pouvoir les découper.

2.1.9 Dynamique basée sur la position

Les approches par éléments finis et particules présentées ci-dessus sont toutes basées sur le calcul de forces agissant sur un corps et sur l'application de ces forces pour déterminer la vitesse et la position des nœuds composant l'objet. La dynamique basée sur la position (PBD), présentée initialement par Müller *et al.* [82], évite complètement le calcul de la force en appliquant des changements directement sur la position des particules. Les différentes forces agissant sur un objet – élasticité, pression, tension, etc. – sont remplacées par des contraintes. En projetant ces contraintes sur les nœuds de l'objet simulé, on obtient un comportement qui obéit aux lois représentées par ces contraintes.

Macklin *et al.* [83] présente une plateforme permettant de construire une simulation interactive à l'aide de la méthode PBD. Une grande variété de matériaux solides, liquides et gazeux peuvent être modélisés en leur appliquant les différentes contraintes. Bender *et al.* [84] dé-

crit plus en détail ces contraintes. L'élasticité peut être simulée principalement à l'aide de contraintes de distance, de volume et d'ajustement de forme (*shape matching*). D'autres contraintes, comme la courbure et la torsion, servent à modéliser le comportement élastique de tissus et de cordes.

Les contraintes de distance agissent comme des ressorts et tendent à maintenir une distance spécifique entre deux particules. Les contraintes de volume tendent à préserver le volume d'un groupe de particules, généralement un tétraèdre. Ces deux types de contraintes font appel à une connectivité entre les particules. À l'opposé, la contrainte d'ajustement de forme pousse un groupe de particules vers sa conformation de repos sans aucune information topologique. Cette méthode, décrite par Müller *et al.* [85], consiste à trouver la combinaison de rotation et de translation qui minimise la somme des carrés des distances entre chaque particule et sa position au repos.

Il est à noter que toutes ces contraintes sont uniquement géométriques et ne reflètent pas la physique des matériaux élastiques. Cette méthode a été développée initialement pour l'animation dans un contexte de production pour films ou jeux vidéos, où l'apparence et la crédibilité sont plus importantes que l'exactitude physique du résultat. Chaque contrainte est associée à un facteur précisant à quelle vitesse la correction est apportée à la position. Ce facteur constitue une mesure partielle de la rigidité du matériau. Cette méthode permet de reproduire de façon crédible le comportement des matériaux sans toutefois être physiquement exacte.

Camara *et al.* [86] implante un simulateur de néphrectomie partielle basé sur la plateforme de Macklin *et al.* [83] en utilisant la contrainte d'ajustement de forme. Les auteurs explorent de façon systématique l'espace des paramètres possibles – taille et nombre de grappes de particules, constante de rigidité, nombre d'itérations du solveur – pour obtenir la déformation la plus proche possible des déformations réelles sur un rein de porc. La méthode d'ajustement de forme ne permet cependant pas de facilement changer la topologie de l'objet étant donné que les grappes de particules utilisées pour l'ajustement de forme se chevauchent.

Pan *et al.* [87] ajoute une contrainte laplacienne [88] à celle de distance pour mieux préserver la distribution locale des nœuds lors de la déformation. Les contraintes sont simplement retirées pour découper. Comme les contraintes de distances sont équivalentes à des ressorts, aucune masse n'est perdue avec l'ajout de coupures.

Bender *et al.* [89] décrit une contrainte PBD basée sur la mécanique des milieux continus, semblable à la méthode des éléments finis, qui tend à minimiser l'énergie de déformation telle que définie par la formulation faible de l'équation modèle 1.1. L'article démontre que, sous certains paramètres, une itération de l'algorithme de projection de cette contrainte est

équivalente à une intégration explicite dans le temps de la méthode par éléments finis décrite par Sifakis et Barbič [3]. Les changements de topologie peuvent être faits simplement en retirant la contrainte d'énergie, ce qui a le même effet que retirer un élément d'un maillage d'éléments finis, mais sans le besoin de mettre à jour une matrice de rigidité.

Pan *et al.* [90] présente un simulateur chirurgical qui introduit une méthode de découpage dans le modèle de Bender *et al.* [89]. La déformation est effectuée avec un mélange de contraintes de préservation de distance, de volume et d'énergie. Le découpage se fait en retirant ces contraintes, en introduisant de nouveaux nœuds et en ajoutant de nouvelles contraintes pour correspondre à la nouvelle topologie. Les tétraèdres sont découpés lorsque la lame du scalpel a complètement dépassé l'arête ou la face où elle a été introduite et les nouveaux éléments sont générés de manière similaire à Mor et Kanade [24].

L'inconvénient principal de la méthode PBD est que la solution à chaque pas de temps dépend d'un solveur qui converge vers un objet complètement rigide. Si on laisse le solveur terminer toutes les itérations nécessaires pour arriver à la solution exacte, le résultat est un objet non déformable qui se déplace. L'aspect élastique est un effet secondaire d'une résolution partielle des contraintes appliquées.

Bouaziz *et al.* [91] présente une méthode de projection de contraintes, mais en décrivant l'énergie de déformation comme une distance avec un espace de contraintes d'énergie nulle. Les auteurs remplacent le solveur utilisé dans la méthode PBD, qui traite les contraintes indépendamment, par un solveur en deux étapes : la première pour projeter les contraintes de façon parallèle et une pour introduire une contrainte globale de conservation de quantité de mouvement. Le résultat converge vers une solution élastique semblable à celle obtenue avec une intégration implicite d'Euler dans le temps.

Qian *et al.* [92] construit un simulateur chirurgical basé sur la dynamique projective [91]. Pour déterminer les éléments à modifier lors de la coupe, les auteurs décrivent un critère qui dépend de la quantité de chaleur transmise aux différents tissus, qui tient compte de leurs propriétés mécaniques et thermiques, afin de simuler de façon réaliste l'utilisation d'un électrocautère.

Peng *et al.* [93] combine les avantages des méthodes basées sur la force et des contraintes PBD en calculant une force élastique avec la méthode PBA et en y ajoutant une contrainte de volume. Cela permet d'obtenir un modèle élastique physiquement réaliste, avec la contrainte qui agit de façon similaire au terme d'énergie supplémentaire parfois ajouté dans les éléments finis pour simuler l'incompressibilité d'un matériau [94]. Des contraintes additionnelles servent à gérer sans effort supplémentaire les collisions entre objets.

2.1.10 Masse-ressort

Un système masse-ressort représente un objet par un ensemble de masses ponctuelles reliées entre elles par des ressorts virtuels [95]. Les forces agissant sur chaque particule sont généralement dictées directement par sa distance avec ses voisins, bien qu'il soit possible de créer des ressorts de torsion et de courbure.

Les systèmes masse-ressort ont été beaucoup utilisés pour la simulation chirurgicale étant donné leur vitesse d'exécution. En effet, la force est donnée par l'élongation des ressorts, sans besoin de calcul de rotation ou d'intégration d'énergie de déformation. De plus, comme les ressorts sont tous indépendants les uns des autres, il est plus simple de paralléliser ces systèmes de façon massive, en utilisant une carte graphique [96, 97].

La principale difficulté concernant l'utilisation de ces systèmes consiste à trouver les paramètres de ressorts qui correspondent aux propriétés mécaniques de l'objet simulé. Certains auteurs font appel à des réseaux de neurones pour chercher l'espace des paramètres possibles pour trouver la meilleure combinaison [98]. D'autres tentent de calculer les paramètres d'après des expériences d'élongation et de cisaillement en fonction du module de Young et du coefficient de Poisson [99]. Cependant, ces paramètres sont valides uniquement dans certaines circonstances, par exemple sur des blocs – ou éléments – cubiques, dans le cas de Baudet *et al.* [100]. Les paramètres choisis sont invalidés lors des changements de topologie et les méthodes utilisées pour les trouver ne sont généralement pas applicables pour déterminer en temps réels les nouveaux paramètres physiquement réalistes.

Jarrousse [101] élabore un système masse-ressort en utilisant les principes de la mécanique des milieux continus. Les particules sont reliées par des ressorts de façon à former des éléments tétraédriques et la force des ressorts est déterminée à chaque pas de temps d'après une fonction de minimisation de l'énergie de déformation des tétraèdres et une contrainte de préservation de volume. Cette approche délaisse cependant les avantages de vitesse d'exécution et de facilité d'implantation des systèmes masse-ressort et se rapproche plutôt des principes des méthodes par éléments finis ou PBD.

2.2 Solveurs

Indépendamment de la manière dont sont calculées les différentes valeurs physiques, comme la position, la force, la déformation ou l'énergie, on se retrouve éventuellement avec un système matriciel à résoudre, comme l'équation 2.1. Différentes méthodes, ou *solveurs*, permettent d'atteindre une solution exacte ou approximative de ce système. Dans ce document, le terme solveur sera utilisé pour décrire les approches pour résoudre trois différents aspects d'une

simulation : l'intégration du système dans le temps, la résolution d'un ensemble de contraintes et la résolution d'un système linéaire tel que $\boldsymbol{Ax} = \boldsymbol{b}$. Plusieurs solveurs peuvent être utilisés au sein d'une même simulation. Une grande variété de solveurs existent pour résoudre des équations aux dérivées partielles ; nous nous concentrons dans cette section uniquement sur ceux qui permettent une simulation en temps réel et qui ont été utilisés avec succès dans des simulateurs chirurgicaux.

2.2.1 Statique

Une approche simple pour gérer la variable de temps consiste à tout simplement l'ignorer et résoudre le système statique de l'équation 2.7, reprise ici :

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}^{ext}. \tag{2.20}$$

Pour les modèles linéaires d'élasticité, la matrice $\boldsymbol{K}$ ne change pas. Il est donc possible de calculer son inverse à l'initialisation de la simulation et de trouver directement les déplacements d'après les forces externes [14, 16, 18, 102] :

$$\boldsymbol{u} = \boldsymbol{K}^{-1}\boldsymbol{f}^{ext}. \tag{2.21}$$

Avec une matrice inverse préalablement calculée, il devient difficile d'effectuer des changements dans la topologie des éléments, étant donné qu'une modification locale de $\boldsymbol{K}$ occasionne des changements plus globaux dans son inverse [18].

L'interaction avec un utilisateur dans une simulation dynamique est par ailleurs compliquée par l'omission de la variable temporelle. Deux options sont envisageables : imposer des déplacements sur les nœuds touchés par l'outil chirurgical et calculer la force résultante sur l'outil [18], ou estimer une force d'après la distance de pénétration de l'outil dans l'objet [102]. Dans les deux cas, les interactions peuvent donner lieu à des déplacements brusques de l'organe simulé étant donné que l'inertie n'est pas prise en compte.

Les principaux avantages d'une solution statique se trouvent dans la facilité d'implantation et la vitesse d'exécution. En particulier, l'utilisation de solveurs linéaires sur GPU permet de simuler un grand nombre d'éléments de manière interactive [102].

2.2.2 Explicite

Un solveur explicite estime la vélocité et la position à la fin d'un pas de temps à partir de l'accélération au début du pas de temps. Par exemple, en insérant l'équation 2.5 dans

l'équation 2.1, on obtient directement les valeurs recherchées :

$$\ddot{\boldsymbol{u}}(t) = \boldsymbol{M}^{-1}\left(\boldsymbol{f}^{ext}(t) - \boldsymbol{K}\boldsymbol{u}(t)\right) \tag{2.22}$$

$$\dot{\boldsymbol{u}}(t+\Delta t) = \Delta t\ddot{\boldsymbol{u}}(t) + \dot{\boldsymbol{u}}(t) \tag{2.23}$$

$$\boldsymbol{u}(t+\Delta t) = \Delta t\dot{\boldsymbol{u}}(t) + \boldsymbol{u}(t). \tag{2.24}$$

Ainsi, une seule évaluation du produit $\boldsymbol{K}\boldsymbol{u}$ peut suffire pour déterminer le nouveau déplacement de l'objet, résultant en un solveur rapide, utilisé tant avec les éléments finis [2, 14, 103–105], les particules [70,71,106] et les systèmes masse-ressort [107]. Des variantes de cette méthode peuvent augmenter la stabilité, par exemple en utilisant la vitesse de la *fin* du pas de temps pour déterminer la nouvelle position [97, 107], en utilisant plusieurs valeurs précédentes [71, 93, 108], ou en utilisant un facteur de relaxation [106, 109] Elle est également fréquemment utilisée sur GPU [96, 97, 109–113].

2.2.3 Implicite

Malgré la simplicité des méthodes explicites, elles ont des contraintes strictes sur la longueur du pas de temps pour garantir leur stabilité. Cette longueur inversement proportionnelle à la rigidité de l'objet et à la taille des éléments qui le forment. Lorsqu'une simulation permet de découper librement les objets déformés, il est donc difficile de poser une limite sur le pas de temps.

Les méthodes implicites garantissent la stabilité de la simulation et permettent des pas de temps arbitrairement longs [5]. La méthode de Baraff et Witkin [114] est fréquemment utilisée comme approximation pour résoudre le système dynamique dans des simulations interactives [115, 116]. En insérant éq. 2.6 dans le système 2.1, en utilisant une approximation de Taylor de premier degré pour estimer le gradient de la force élastique $\boldsymbol{K}\boldsymbol{u}$ et en ignorant l'amortissement, on obtient

$$\left(\boldsymbol{M} - \Delta t^2\boldsymbol{K}\right)\Delta\dot{\boldsymbol{u}} = \Delta t\left(\boldsymbol{f}^{ext} + \boldsymbol{f}^{int}\right). \tag{2.25}$$

Une dérivation plus complète de cette équation est disponible en annexe B.

Cette méthode d'intégration est fréquemment utilisée en combinaison avec un solveur linéaire itératif sans matrice, comme le gradient conjugué [25], qui n'assemble jamais explicitement la matrice $\boldsymbol{K}$ et ne nécessite donc aucun effort additionnel lorsque la topologie de l'objet est modifiée. Elle a été utilisée sur GPU [117,118], incluant des comportements non linéaires [119].

Multigrille

Afin d'accélérer la convergence des méthodes itératives, il est possible d'utiliser des solveurs multigrille, qui résolvent le système successivement à des résolutions différentes et parviennent à une solution en moins d'itérations que le gradient conjugué. Des solveurs multigrille ont servi à simuler des éléments finis cubiques multirésolution avec découpage [42] et ont été implantés sur GPU [41].

2.2.4 Mixte

Certaines méthodes de simulation sont dites de "prédiction-correction", qui fonctionnent en deux étape. L'étape de prédiction détermine d'abord la position des nœuds d'un objet ou d'un ensemble d'objets d'après leur vélocité. Un solveur explicite est généralement utilisé pour cette étape [82]. L'étape de correction calcule ensuite la position plus exacte des nœuds en tenant compte des contacts avec les autres objets de la scène ainsi que d'autres contraintes.

La méthode de *compliance* [120], décrite plus en détail dans la section 2.4, calcule précisément les forces de contact entre objets pour empêcher leur interpénétration [115]. Elle permet également de calculer les forces de friction entre objets [116].

L'autre cas important d'utilisation d'une méthode de prédiction-correction est avec la méthode PBD. Une fois les particules déplacées, un solveur de Gauss-Seidel est utilisé pour résoudre les différentes contraintes imposées sur le système [82, 83, 86, 90].

2.3 Rendu visuel

Deux types de méthodes sont principalement utilisées pour afficher les objets simulés : une représentation directe de la surface implicite d'après les particules ou un maillage généré à l'avance qui se déforme avec l'objet. Initialement, avec les éléments finis, les faces externes des éléments à la frontière pouvaient être affichées directement. Cependant, avec l'évolution des méthodes de découpage et des capacités graphiques des ordinateurs, il est devenu nécessaire de séparer l'apparence d'un objet de son modèle physique sous-jacent [21,40,41,104,121,122].

Avec des systèmes de particules, la surface n'est pas définie explicitement. L'affichage d'un objet peut donc se faire directement avec une technique qui génère implicitement une surface sur l'objet, comme le lancer de rayon [123], le *splatting* [79] ou des surfaces de niveaux [124]. Plusieurs variantes de ces méthodes sont utilisées dans le domaine de l'animation [57, 61], où les applications ne nécessitent pas un traitement en temps réel.

Le *splatting* est utilisé par De *et al.* [79] dans un simulateur de chirurgie interactif, avec une

résolution relativement faible. Luo *et al.* [125] combine le *splatting* avec une surface explicite pour obtenir une performance supérieure sans sacrifier la qualité du rendu visuel.

Zou *et al.* [126] produit une image réaliste en utilisant des captures vidéo pour extraire une texture de l'organe simulé. Cette texture est ensuite plaquée sur le modèle déformé. Cette méthode nécessite cependant l'accès à un enregistrement de l'opération chirurgicale qu'on désire simuler.

Cependant, pour obtenir un rendu plus raffiné sans utiliser trop de ressources, une surface explicite formée de triangles sert généralement à représenter visuellement l'objet simulé. La même approche de découplage entre les modèles physique et visuel qu'avec les objets simulés avec maillage est alors préconisée [66, 67, 69, 73, 90, 92, 127].

Lorsqu'une surface explicitement triangulée est utilisée, le maillage de surface est généralement rattaché au modèle physique en utilisant les coordonnées barycentriques des éléments ou en utilisant les positions et les normales des particules. Lors de changements de topologie, les triangles de la surface sont découpés et de nouveaux triangles sont ajoutés le long de la surface exposée par la coupure. Une triangulation de Delaunay [128] permet par exemple de générer dynamiquement un nombre minimal de triangles bien formés. Lorsque la coupure est représentée par une séquence de plans en forme de parallélogramme, ils peuvent simplement être divisés en deux triangles [67].

2.4 Rendu haptique

Le rendu haptique consiste à produire une force sur l'appareil haptique qui correspond à la force transmise à l'outil interactif virtuel par les différents éléments de la scène simulée. Cette force doit correspondre aux mouvements de l'utilisateur et à ses interactions avec l'environnement virtuel. Le calcul de cette force est relié de près au calcul du contact entre les objets simulés.

Lorsque la déformation est modélisée par un problème quasi-statique, l'outil simulé prescrit généralement un déplacement sur certains nœuds [79, 102]. Les nœuds de l'objet peuvent être séparés en ceux qui ont une position établie par le contact l'outil – les nœuds *fixés* – et ceux qui se déplacent sous l'effet des forces élastiques [79] – les nœuds *libres*. Le système matriciel de 2.7 devient

$$\boldsymbol{K}\boldsymbol{u} = \begin{bmatrix} \boldsymbol{K}_{aa} & \boldsymbol{K}_{ab} \\ \boldsymbol{K}_{ba} & \boldsymbol{K}_{bb} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_{tool} \\ \boldsymbol{u}_b \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{tool} \\ \boldsymbol{f}^{ext} \end{bmatrix} = \boldsymbol{f}. \tag{2.26}$$

Cette séparation donne lieu à deux sous-problèmes qui permettent de trouver les positions

des nœuds libres :

$$\boldsymbol{u}_b = -\boldsymbol{K}_{bb}^{-1}\boldsymbol{K}_{ba}\boldsymbol{u}_{tool} \tag{2.27}$$

ainsi que la force que les nœuds fixés exercent sur l'outil :

$$\boldsymbol{f}_{tool} = \boldsymbol{K}_{aa}\boldsymbol{u}_{tool} + \boldsymbol{K}_{ab}\boldsymbol{u}_b. \tag{2.28}$$

Dans une simulation dynamique, deux méthodes générales sont utilisées pour modéliser les contacts. La méthode plus complexe, qui tient compte de tous les contacts possibles entre les différents objets de l'environnement consiste à modéliser une force de contact $\boldsymbol{\lambda}$ et à la calculer en résolvant un problème de complémentarité linéaire (LCP) [120] :

$$\boldsymbol{\delta} = \boldsymbol{H}\boldsymbol{C}\boldsymbol{H}^T\boldsymbol{\lambda} + \boldsymbol{\delta}_{free}, \tag{2.29}$$

où $\boldsymbol{\delta}$ est un vecteur contenant les distances de pénétration permises à chaque point de contact, $\boldsymbol{\delta}_{free}$ aux distances trouvées lorsque les objets bougent librement, sans entrer en collision les uns avec les autres, $\boldsymbol{H}$ une matrice reliant le déplacement à chaque paire de points en contact à leur distance de pénétration et $\boldsymbol{C}^{-1}$ une matrice dépendant de $\boldsymbol{M}$ et de $\boldsymbol{K}$ (équation 2.1). $\boldsymbol{C}$ est appelée matrice de *compliance*. Une description de la méthode pour obtenir les matrices $\boldsymbol{C}$ et $\boldsymbol{H}$ se trouve dans l'annexe C.

Cette approche est utilisée entre autres par Courtecuisse *et al.* [26] et Jung *et al.* [129] pour donner directement les forces ressenties par l'outil durant les manipulations par l'utilisateur.

La deuxième méthode de modélisation de contacts consiste à appliquer une force à partir de l'outil simulé sur les objets et calculer leur déformation plutôt que de les déformer et calculer la force résultante. La force appliquée par l'outil est déterminée par une méthode dite de *pénalité* : la distance d'interpénétration entre l'outil et l'objet définit un ressort virtuelle de longueur zéro au repos contenant donc une certaine énergie si la distance est non-nulle, qui est traduite en force [130]. Cette approche est utilisée par Barbič et James [131] et Pan *et al.* [132]. Cirio *et al.* [133] utilise une méthode similaire dans un milieu simulé par particules, mais en modélisant également l'outil avec des particules entourant un cœur rigide. Cette façon de faire permet de simuler l'outil comme n'importe quel autre objet de la scène et même d'avoir un outil déformable.

Afin d'obtenir un rendu haptique stable, il est nécessaire de mettre à jour la force du dispositif haptique à une haute fréquence, généralement plus de 300 Hz. Plus la rigidité des objets simulés est grande, plus cette fréquence doit être élevée pour que le rendu soit à la fois stable et suffisamment rigide [130]. Généralement, les calculs de déformation et de collisions se font

trop lentement pour rafraîchir le rendu haptique directement. Deux options sont possibles pour contourner ce problème. Le calcul de force peut se faire à partir d'un modèle simplifié de collision ou de déformation afin d'atteindre le taux désiré – pendant que les calculs détaillés se font à une fréquence plus réduite. La force retournée peut aussi être extrapolée à partir de la force courante et de sa vitesse de variation entre deux étapes de calcul de la physique. Une manière courante de stabiliser le retour de force consiste à coupler virtuellement l'outil simulé – dans l'environnement virtuel – à la position du dispositif haptique tel que tenu par l'utilisateur [130, 134].

2.5 Objectifs de recherche

L'objectif principal de cette recherche est de développer une méthode physique de simulation de corps déformables, dont la topologie peut être efficacement modifiée afin de simuler leur découpage. Cette méthode permettra de créer un simulateur de chirurgie ouverte dont le principal moyen d'interaction avec l'utilisateur est à l'aide d'un dispositif de retour d'effort. Le rendu visuel, en combinaison avec le rendu haptique, doit permettre de recréer un environnement crédible pour l'utilisateur.

2.5.1 Hypothèses

H1 Une méthode avec particules permet de décrire la déformation d'un corps sous l'effet de forces externes avec une erreur inférieure à 10% par rapport à une méthode avec éléments finis.

H2 Une méthode avec particules permet de calculer en temps réel (au moins 30 Hz) la déformation d'un corps discrétisé à une résolution de 10000 nœuds.

H3 Une méthode avec particules permet de modifier rapidement et aisément la topologie de l'objet simulé, avec un temps de calcul inférieur à 10% du temps de calcul total à chaque pas de temps.

2.5.2 Objectifs spécifiques

Pour vérifier les hypothèses ci-dessus, les objectifs suivants ont été définis :

O1 Implanter une méthode interactive de déformation avec particules. (**H1**, **H2**, **H3**)

O2 Modifier la topologie du volume et de la surface d'un système de particules, tout en préservant le mappage entre les deux. (**H3**)

O3 Implanter la méthode de déformation, découpage et génération de surface pour qu'elle s'exécute efficacement sur une carte graphique. (**H2**)

CHAPITRE 3 MÉTHODE

3.1 Contexte du projet

Le chapitre 2 a présenté les différentes méthodes de simulation existantes qui permettent de modéliser le découpage d'objets déformables avec un outil tranchant.

Plusieurs de ces méthodes simulent fidèlement un découpage arbitraire. Elles ont toutes comme désavantage soit le temps de calcul élevé pour apporter les changements, soit une grande complexité qui limite la quantité ou la vitesse à laquelle les coupures peuvent être ajoutées au modèle. Les autres méthodes discutées font plusieurs concessions quant à la qualité de la description des coupures, ou celle des déformations calculées à la suite de ces coupures.

On cherche donc lors de ce projet à développer une méthode de simulation des corps mous qui répond aux critères présentés ci-dessous.

3.1.1 Déformation physiquement réaliste

Un calcul de la déformation basé sur la mécanique des milieux continus donne les résultats les plus réalistes. Elle permet de facilement ajuster les paramètres physiques des matériaux simulés, comme le module de Young, le coefficient de Poisson et la densité.

3.1.2 Coupe libre, exacte et progressive

L'utilisateur doit pouvoir appliquer des coupures librement, contrairement aux méthodes qui prédéfinissent des lignes de coupe. En retour, les coupures qui apparaissent doivent suivre exactement le mouvement de l'outil plutôt que s'ajuster aux éléments de la discrétisation. L'aspect visuel d'une coupure durant l'incision doit correspondre exactement à la position de l'outil en mouvement. De plus, la qualité de la simulation de déformation ne doit pas être affectée par l'ajout de coupures.

3.1.3 Faible coût de la simulation des coupures

Le calcul de déformation physiquement réaliste demande une grande quantité de ressources. Les changements topologiques arbitraires des méthodes existantes augmentent souvent de façon importante le temps de calcul, en particulier les méthodes avec maillage. La méthode

désirée utilisera seulement une faible proportion du temps de calcul total pour effectuer le découpage. Par ailleurs, l'ajout constant de coupures ne causera pas de dégradation progressive de la performance.

3.1.4 Interactivité

Étant donné que la méthode développée doit s'insérer dans un simulateur existant – présenté dans la section 3.2 – elle doit permettre le calcul d'un retour d'effort et afficher à haute résolution les objets simulés. Elle permettra également de modéliser des outils de forme arbitraire et de les utiliser pour ajouter des coupures aux modèles déformables simulés.

3.1.5 Implantation GPU

Lorsque la déformation est calculée sur GPU, il est généralement possible d'augmenter la résolution des objets simulés. Les changements de topologies dans les méthodes existantes demandent souvent le transfert d'une grande quantité de données ou nécessitent une proportion importante du temps de calcul. La méthode désirée doit pouvoir s'exécuter sur GPU afin de simuler un grand nombre de nœuds, sans toutefois faire augmenter le temps pris par l'insertion de coupures.

3.2 Environnement

3.2.1 Simulateur chirurgical

Le travail présenté dans ce document s'inscrit dans une collaboration avec OSSimTech, avec l'intention d'incorporer la méthode développée dans un simulateur chirurgical complet. Ce simulateur comprend un environnement en réalité virtuelle qui fournit un retour haptique sur des objets rigides et un rendu visuel en 3D.

Un tel simulateur inclut beaucoup plus de fonctionnalités que celles décrites au chapitre 2. Pour générer un environnement plus complet, il simule différents aspects physiques, détecte les collisions entre tous les éléments, calcule un rendu visuel global réaliste, gère divers scénarios d'opération chirurgicale et guide et évalue l'utilisateur lors de sa pratique simulée.

Les aspects les plus pertinents en lien avec l'intégration de ce projet dans un simulateur sont les autres éléments physiques, la détection de collisions – reliée de près à l'interactivité et au retour d'effort – et le rendu visuel.

Les autres éléments physiques sont les corps rigides et les fluides. Les corps rigides comprennent les os, qui peuvent être percés, sciés ou grugés lors de manipulations, ainsi que des

objets présents dans la scène, mais qui ne peuvent être manipulés, comme la table d'opération. La simulation de ces aspects spécifiques est presque indépendante de la simulation des corps mous, excepté pour le fait qu'elle peut demander une quantité de ressources considérable, qui doivent être partagées, et pour les interactions entre les deux types de corps.

Les interactions mêmes sont déterminées par les collisions entre objets. Le problème de détection générale des collisions n'est pas traité directement dans ce projet. Cependant, étant donné le besoin d'avoir des interactions précises avec l'outil, une représentation explicite de la surface est la plus appropriée, tant pour la détection des collisions avec un outil chirurgical et les autres objets que pour le rendu visuel. Une surface triangulée pour représenter tous les objets permet de les combiner tous dans la même méthode de détection des collisions.

Le réalisme du rendu visuel de la scène entière augmente avec différents effets ajoutés, comme les textures, les ombres, les lumières dynamiques, les reflets texturés, etc. Cependant, chaque effet demande des ressources additionnelles, à partager avec les autres aspects de la simulation. De plus, pour obtenir une vision 3D réaliste, il est nécessaire d'effectuer le rendu deux fois par rafraîchissement de la simulation – une fois pour chaque œil – ce qui double les calculs nécessaires.

Pour augmenter les capacités du simulateur, deux cartes graphiques sont utilisées : une dédiée au rendu visuel, et l'autre pour les calculs de collisions et de physique. Ainsi, la méthode développée lors de ce projet s'exécutera sur un GPU partagé avec d'autres simulations physiques.

3.2.2 Développement

Ce projet sera développé indépendamment du simulateur décrit ci-dessous, afin de bien tester la méthode en isolation. Nous utiliserons à cet effet la plateforme logicielle SOFA [115], qui permet de résoudre différentes variantes du problème d'élasticité pour des applications médicales. Elle utilise des fichiers simples pour décrire les objets de l'environnement simulé, où les différents paramètres comme le tenseur de déformation, la loi de comportement, la méthode de discrétisation ou la méthode d'intégration dans le temps peuvent être facilement échangés. Il sera ainsi aisé de comparer notre méthode à celles déjà implantées et de tester différentes approches.

3.3 Axes de recherche

Trois aspects essentiels de la simulation de découpage de corps déformable sont explorés dans les prochains chapitres : une méthode de déformation réaliste, qui demeure valide lorsque la

topologie initiale est modifiée ; l'interaction de cette méthode en présence d'un outil ; et la mise à l'échelle de la méthode pour des problèmes de plus grande taille grâce à l'utilisation d'un GPU.

3.3.1 Real-time visual and physical cutting of a meshless model deformed on a background grid

Cet article présente une structure composée pour décrire la topologie d'un objet pouvant être découpé, ainsi que l'algorithme associé pour effectuer le découpage. La déformation est calculée à l'aide de la méthode EFG, qui assure une simulation stable et définit un volume précis. L'article présente également un algorithme pour modifier progressivement et exactement la surface d'après la progression d'un outil modélisé par un plan de découpe. Nous évaluons la performance des deux algorithmes par rapport au temps de calcul de déformation pour démontrer l'efficacité de cette méthode de découpage.

3.3.2 Dynamic cutting of a meshless model for interactive surgery simulation

Dans cet article, nous combinons les algorithmes de découpage avec le mouvement de l'outil et le calcul de déformation. Un soin particulier doit être porté à ces interactions, étant donné que le mouvement de l'outil provoque des changements dans la position de l'objet et sa déformation, mais ces changements ne doivent pas causer d'intersections additionnelles. Nous décrivons également une méthode pour recalculer l'association de la surface avec le modèle volumique lors de la progression d'une coupure. Nous présentons ensuite différents scénarios dynamiques qui démontrent l'interaction entre l'outil et un corps déformable lorsque les deux sont en mouvement.

3.3.3 GPU-friendly data structures for real time simulation

Dans cet article, nous décrivons une structure pour représenter les différentes connectivités du premier article (chapitre 4). Cette structure permet une modification efficace de la topologie en minimisant la quantité de données à modifier et à transférer entre l'hôte et le GPU. Des tests de performance sont effectués et démontrent l'efficacité de la méthode par rapport à une implantation sur CPU ainsi que le maintien de la performance du découpage par rapport au reste de la simulation.

CHAPITRE 4 REAL-TIME VISUAL AND PHYSICAL CUTTING OF A MESHLESS MODEL DEFORMED ON A BACKGROUND GRID

Authors: Vincent Magnoux and Benoît Ozell

Published in *Computer Animation and Virtual Worlds*, June 2020.

DOI: 10.1002/cav.1929

Abstract

Soft body deformation models are commonly used in surgery simulations. However, cutting those models can have a severe impact on computation times and affects the interactivity of the simulation. We propose a novel method for modeling topology and introducing cuts in a meshless soft body simulated on a background grid, as well a way to progressively update the visual aspect of the object by adding a small number of triangles to the surface mesh to cover the cut area. We determine that the accuracy of the deformation is preserved after cutting by comparing our method to a finite element method. Tests show that this new method achieves interactive simulation rates with more than 10000 elements while cutting the model and reconstructing the mesh. Our separation of the visual and physical aspects of the simulation allows for more flexibility when tuning the performance of the simulation. Topology modifications have little impact on computation times for either physical or visual changes.

4.1 Introduction

Cutting deformable objects poses some challenging problems when we want to use them for surgery simulation. These simulations need to meet many criteria so that a user can interact with the virtual environment in a way that is credible and will also allow them to learn the gestures to perform during a surgical operation. The simulation needs to be fast to be interactive, it must be accurate to closely mimic the behavior of organs during surgery and it must be sensed visually and haptically so that users perceive the effect of their actions.

The necessity for speed arises in all steps of the simulation. It is needed to quickly transmit user gestures to the virtual world, to detect and react to contacts between a user-manipulated tool and other virtual objects, to determine how these objects are displaced, deformed and otherwise modified and to convey all these changes and interactions back to the user. Simul-

taneously, all steps require a certain level of accuracy: the tool should be positioned where the user puts it, contacts should be detected exactly where they occur and should produce the proper forces, the displacement of organs and other simulation elements should obey to the laws of physics, and the user should see where the virtual objects are and should feel the force as if they were holding an actual surgical tool.

Simulations often strive for the best balance between the speed and accuracy aspects, since better accuracy usually requires more computation time, for a given method. For the purpose of surgery simulation, models based on continuum mechanics are most often used for their good accuracy. They rely on a large amount of precomputation to achieve sufficient speed. However, when a model is cut, as is needed to simulate an incision or organ resection, some of these precomputed values become invalid and may be very costly to update in such a simulation.

4.1.1 Related Work

Simulation methods for the behavior of deformable objects are often classified according to their dependence on the use of a mesh. At one end of the spectrum, finite element methods (FEM) [14] and mass-spring systems (MSS) [11] depend on an explicitly specified mesh that completely defines the topology of the object. The vertices of the mesh correspond to degrees of freedom (DOF) at which the system is solved; they are bound in pairs for MSS or in groups of four or more when using polyhedral elements in FEM. At the other end, some particle-based methods do not contain a definite topology and each material point of the simulated object only depends on the proximity and distribution of particles around it. The particles correspond to the degrees of freedom and are bound in groups according to their proximity to each other, for example in point-based animation [57] or in smoothed particle hydrodynamics (SPH) [61]. In between, there are methods that rely on both a mesh and a set of particles, or on either, depending on how an object is specified. In element-free Galerkin methods (EFG) [56], a background mesh of elements is used for integration and particles are bound to an integration point according to their distance from it. In position-based dynamics (PBD) [82] and projective dynamics [91], particles may be bound in pairs by distance constraints, in tetrahedra by volume and energy constraints, or in other configurations to simulate different kinds of materials. The various constraints allow these methods to behave like MSS, FEM, SPH or a mix of these. This review is structured according to the way topology information is encoded into the simulated object rather than the method used to compute deformation.

Volume Cutting

Early implementations for real-time simulation would use the same mesh for solving soft body deformations, for encoding topology and for rendering [14]. This restriction poses strict requirements on the capability of mesh to properly represent details of both the surface and of introduced cuts as well as accurately capture interactions through the surface. A common strategy for reducing the requirement on the resolution of the solver mesh consists in separating the *surface representation* from the *deformation mesh* [20, 21, 42, 122]. This means that the deformation mesh only needs to approximate the geometry of the object.

To represent cuts physically, the deformation mesh needs to be modified in some way. Some authors use node snapping [12] to accurately follow the path of the cut while preserving the number of elements in the mesh, thus limiting the number of changes to the system matrix that many fast linear methods end up using. The downside is that elements adjacent to the cut are deformed and may degrade the quality of the mesh [7]. Others divide the elements themselves along the cut and progressively update the system matrix [9, 26, 27]. This allows for a very accurate representation of the cut at a relatively low computational cost. However, it requires careful, structured updates to the matrix and degrades the performance over time, with every new cut introduced into the model. This degradation may be corrected periodically by computing a new stiffness matrix in a separate thread as the simulation proceeds [9, 26].

Another way to avoid potentially difficult modifications to the simulation mesh is to enrich the shape functions with additional degrees of freedom to model discontinuities rather than explicitly modify the topology [45,47,49,50]. The computational relationship between existing degrees of freedom is not affected since the cuts are simply superimposed on the initial model. It is also possible to use this technique with meshless methods [70]. While they allow for a very accurate cut representation at a low computational cost, these methods are not well suited to describe multiple cuts that are in contact with each other.

To avoid altogether numerical issues that relate to element shapes, some authors use the virtual node technique [6,34], where a cut element is rather duplicated and the nodes of each separate part are assigned properties according to how these parts are filled with material. Others go further into separating topology representation from the physics mesh by using voxels to encode material topology [21] or by specifically using a separate mesh for the material [20]. The main advantage of these methods is that the FEM mesh quality is not compromised by topology changes and performance is not affected in a major way.

Some authors organize hexahedral finite elements hierarchically in an octree structure [20, 42, 43]. When cutting, this organization offers a simple way to arbitrarily refine elements to

the desired resolution and to properly distribute their material properties. The drawback of that method is that it increases the number of simulated nodes as cuts are added to the model and the hierarchy is refined.

Meshless methods do not necessarily rely on a mesh for describing the geometry of an object. However, a topology does exist, even if only implicitly in the forces that bind the degrees of freedom together. Methods such as SPH or point-based animation allow for topology changes based on material properties like temperature by loosening the forces between particles when they are heated and producing effects like melting. However, sharper separations such as cuts are more difficult to simulate without a specific representation of the topology. Most cutting approaches rely on determining the proper neighborhood of each particle so that the influence between material points is determined by their *topological distance* rather than their Euclidean distance. The ones that achieve real-time interactions are based on a visibility criterion implemented either with a graph that allows for quick calculation of topological distance between particles [10, 73] or with the use of level sets to determine whether pairs of particles see each other [75].

In PBD and projective dynamics, topology is encoded in the constraints applied to the particles. Simply using distance constraints between pairs of particles will result in something very similar to a mass-spring system, but more complex constraints like tetrahedron volume or energy may give the same results as finite elements [89]. With distance constraints, cutting is simply a matter of removing the constraints so that particles on either side of the cut no longer influence each other. With more complex constraints, if the volume must be preserved, new constraints must be added and the methods for topology modifications are very similar to those explained above for tetrahedron cutting. They have successfully been applied in surgery simulation settings [90, 135], sometimes mixing them with other deformation methods [87].

Surface Cutting

Cutting the surface of an object is somewhat simpler, since there is one fewer dimension to account for in the surface topology. Numerical requirements are also more relaxed on the surface mesh because it does not affect the result of the deformation, although the surface shape may have some impact on the performance of collision detection. In some cases, when the surface is automatically generated from the topology of the object, cutting it may not require a specific additional effort compared to cutting the volume.

Methods for surface generation include marching cubes [21], splitting cubes [136] and dual contouring [121]. The explicit surface triangulation is usually calculated before the simulation and updated locally when a cut is introduced in the model. New triangles are added where

topological links are removed. Using these methods, the resulting surface has the same resolution as the topology description, but can be smoothed to closely match the trajectory of the cut [21].

When the surface is specified independently of the object topology, it also has to be cut separately. This cutting operation can be divided in two steps: splitting the existing surface along the cut and creating a new surface on the parts of the object that have been exposed by the cut. The first step can be done by dividing every triangle intersected by the cutting tool in three new subtriangles [10] or by snapping the closest vertex of intersected triangles on the cut and separating them along their edges [13]. For the second step, a new surface may be defined using a Delaunay triangulation method [127] or the mesh of the cutting tool itself [10]. Both steps can be combined by considering the triangles of both the cutting tool and the cut object as one "triangle soup" [34], dividing all of them at their intersections and discarding every triangle that does not lie on the surface of a tetrahedron used for volume deformation.

4.1.2 Contributions

This paper presents a simulation method where the deformation is modeled using an EFG method. That method was chosen for its stability, precision and the fact that it is mostly meshless, which makes its topology more simply modifiable. Interactions with the environment and user are modeled through an explicit surface that follows the displacements of the degrees of freedom and transmits contact forces to them.

We describe the following contributions:

— A real-time algorithm for cutting an object modeled with an EFG method, using a topology graph connecting both the background grid and the set of degrees of freedom (DOFs).

— A fast and fully progressive method for meshing the cut area that reuses existing triangles when possible and that always stays consistent with the newly cut existing surface.

Section 4.2 will present the method, while section 4.3 will describe the tests, the results, and discuss the advantages and drawbacks of the method.

4.2 Method

A virtual object is composed of four major structures that each hold characteristics or properties of the object. The degrees of freedom, or *particles*, determine the position of material

points and are the locations where elastic forces are computed and external forces are applied. The *integration grid* describes the volume of the object and allows to accurately perform integration over that volume. The *topology graph* represents the topology of the object, how the particles are connected to each other and to the integration grid. The *surface mesh* corresponds to the geometry of the object; it provides a visual representation as well as an interface between the object and other elements of the simulation, such as a surgical tool.

We first briefly outline the problem to solve before describing how the object is discretized, how its topology is first determined, how it can be changed and how the surface is modified to reflect these changes.

4.2.1 Deformation Model

We seek to solve a dynamic elasticity problem

$$\rho \ddot{\boldsymbol{u}} = \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f}^{ext}, \tag{4.1}$$

where ρ represents mass density, $\boldsymbol{u}$ the displacement, $\boldsymbol{\sigma}$ the stress tensor and $\boldsymbol{f}^{ext}$ external forces. This problem can be discretized and linearized to obtain a system of equations

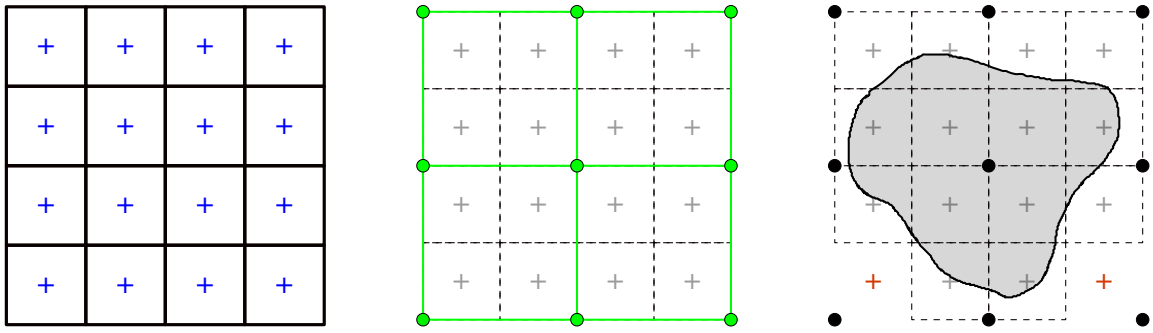
$$\boldsymbol{M}\ddot{\boldsymbol{u}} = \boldsymbol{K}\boldsymbol{u} + \boldsymbol{f}, \tag{4.2}$$

where $\boldsymbol{K}\boldsymbol{u}$ is a vector representing the internal elastic force on each DOF, $\boldsymbol{f}$ the external forces acting on these DOFs and $\boldsymbol{M}$ the mass matrix of the system.

4.2.2 Discretization

The object is discretized as a set of particles which are not directly linked together for the purpose of deformation computation. These computations are rather done over a background integration grid, according to the EFG method [56]. This grid is a regular mesh composed of cubes with a single integration point in their center, illustrated in 2D in Figure 4.1a. Particles may be placed somewhat arbitrarily in the simulated object, as long as they are positioned in a sufficiently regular manner and approximate closely enough the geometry of the object, as shown by Horton *et al.* [71] In our case, we decided to place them in another regular grid that is superimposed on the background grid, as illustrated in Figure 4.1b. The cubes of that DOF grid are called *cells* and have a side length twice that of the integration elements.

This placement is very simple to implement, as it is entirely independent of the shape of the simulated object. This can work because each integration element is weighed according



(a) (b) (c)

Figure 4.1 Two-dimensional representation of the grids used to compute deformation. a) The background integration grid; each square is an integration element, with a corresponding integration point in its center (blue crosses). b) The degrees of freedom (green dots) of the system are placed in a larger grid that coincides with the background grid. In 2D, each group of four DOFs form a cell (green squares). c) The volume (or surface when in 2D) of the integration elements is determined by the proportion that is covered by the object; in this particular example geometry, the two bottom corner integration points (in orange) are discarded.

the proportion of its volume that lies inside the object, as explained below. It also makes it easier to form a stable neighborhood when cutting since we already know whether particles are in the same plane, an important factor when computing shape functions. Finally, using this placement scheme ensures that the ratio of integration point to degree of freedom is high enough to obtain a stable simulation [71].

When initializing an object, the volume of each integration element is estimated as the amount of volume within the element that lies inside the geometry of the modeled object. This is done by sampling a certain number of points in each element and verifying whether they are located inside the object [72]. Only elements with non-zero volume are taken into account during the simulation (see Figure 4.1c). Using this method, even if some particles are located outside the geometry of the object, the simulated deformation remains accurate, as shown in Brunet *et al.* [72].

For each integration element with a non-zero volume, a fixed number of neighboring particles are chosen based on their proximity to the integration point. For a given neighborhood, the shape function of each particle at the integration point is computed with MLS, along with its derivatives. As explained by Belytschko *et al.* [56], the shape function for particle i at integration point I is given by

$$\phi_i(\boldsymbol{x}_I) = \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B}_{Ii}, \tag{4.3}$$

where

$$\boldsymbol{p}^T = \begin{bmatrix} 1 & x & y & z \end{bmatrix} \tag{4.4}$$

$$\boldsymbol{A}_I = \sum_i^{N_I} w_I(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i)\boldsymbol{p}^T(\boldsymbol{x}_i) \tag{4.5}$$

$$\boldsymbol{B}_{Ii} = w_I(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i) \tag{4.6}$$

and a quartic spline function chosen for the kernel:

$$w_I(\boldsymbol{x}) = \frac{105}{16\pi}\left(\frac{1}{h^3} - \frac{6r^2}{h^5} + \frac{r^3}{h^6} - \frac{3r^4}{h^7}\right), \tag{4.7}$$

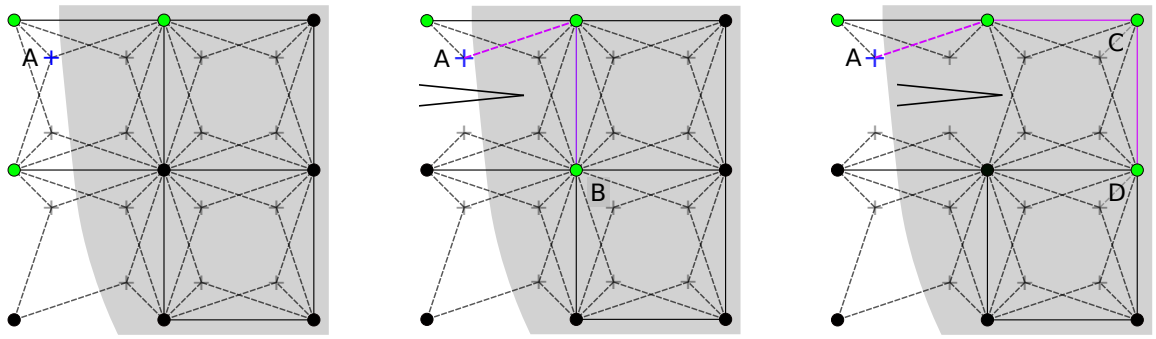
where r is the distance between $\boldsymbol{x}_I$ and $\boldsymbol{x}$, and h is the radius of the region of influence of the integration point multiplied by a constant set to 3 in our case (see Horton *et al.* [71] for a discussion on that parameter). Equation 4.3 uses the inverse of matrix $\boldsymbol{A}$. For that matrix to be invertible, the particles composing the neighborhood of the integration point must not be coplanar (or collinear, in 2D).

Once the volume has been determined, the mass associated with each integration element is computed based on the density of the material. This mass is distributed to the neighbors of the element according to the weights given by the shape functions around that element. This process provides the simulation with a lumped mass matrix to be used in equation 4.2.

4.2.3 Object Topology

The topology of the object is described by the combination of the background grid and the set of particles. The regular grid on which the particles are located provides a simple way to link them together: each particle is linked to its nearest neighbor along each axis in both directions. The entire connectivity graph is composed of the initial links between integration points and particles and of the links between particles. It may be used as a visibility graph in a way very similar to what is described by Jung and Lee [73]. Figure 4.2 shows how neighbors of an integration point are chosen before and after a cut has been introduced into an object.

For the purpose of cutting in three dimensions, the tool blade may be modeled as a line, the edge, which moves as the user displaces it at each frame. The line endpoints in two different animation frames form a quadrilateral that is approximated by two triangles; that quadrilateral is flat when there is no twisting movement from the tool between frames. Any link between particles or between a particle and an integration point that is intersected by that cutting plane is severed, and a new neighborhood has to be formed for each affected



(a) (b) (c)

Figure 4.2 Choosing neighbors during volume cutting. Solid lines indicate links between two DOFs and dashed lines indicate links between a DOF and an integration point. a) Initially, neighbors are chosen only by proximity. b) One particle is cut off integration point A; a new neighbor B is chosen for that point by following the links between particles (in purple) and selecting the closest one which does not result in a collinear set of neighbors. c) The third neighbor is cut off the integration point; since the next topologically closest particle (C) makes the neighborhood collinear, we insert an additional neighbor D to be able to generate valid shape functions.

integration point. To choose the closest particles, we distinguish two neighbor types: direct and indirect. Direct neighbors of an integration point are those that border the DOF cell in which the integration element is located. Indirect neighbors are those found by following links along the topology graph. The distance used to compute shape functions is simply the distance along the graph rather than the Euclidean distance between integration points and their neighbors. This method of determining whether two points are neighbors is sometimes called the visibility criterion, and the topology graph a visibility graph [10].

Once the new neighborhoods have been determined, the shape functions at each modified integration point are recalculated and the mass associated with these points is redistributed to the new neighbors, resulting in a new system to be solved. All updates, including the search for new neighbors and distance computations, are done in the reference, undeformed shape of the object, according to a total Lagrangian formulation [71].

4.2.4 Surface

An explicitly defined surface composed of triangles provides a simple and efficient way to visualize the object modeled by the system of integration points and particles. It also allows to accurately detect contacts with the object and transmit boundary forces to and from it, such as those that occur when a user-manipulated tool touches and cuts the object.

The vertices of the surface mesh are mapped to the particles using MLS shape functions, as in eq. 4.3. The displacement of each vertex is a weighted average of the displacement of its neighboring particles, with the weights determined by the MLS functions. For the surface to stay consistent with the topology of the object, the position of each vertex must only depend on particles that influence the material point represented by that vertex. To ensure that condition, each vertex is mapped to the same particles as a certain integration point. Initially, the integration point that is nearest to each vertex is chosen.

Forces are transmitted from the surface mesh to the particles through the same mapping. For example, a pressure force acting on a certain area would be distributed to the vertices of the triangles affected by that force. It would then be distributed from each vertex to its neighboring particles with the same weights as for the position.

This mapping allows connecting the surface to the volume of the object while being able to treat them completely separately. This means that the volume can be deformed and cut using any method, and the surface can be modeled using any method, as long as the mapping between the two can be updated in a consistent way.

To modify the appearance of the surface mesh, the same cutting plane as for the volume is used. As with volume cutting, intersections are detected in the deformed shape, but all modifications are done on the initial mesh configuration. New vertices are added at the intersection of the cutting plane with every triangle edge of the existing surface mesh. All triangles that are cut in two are removed and replaced by three new ones, while those that are only partially cut are replaced by four new ones, as illustrated in Figure 4.3.

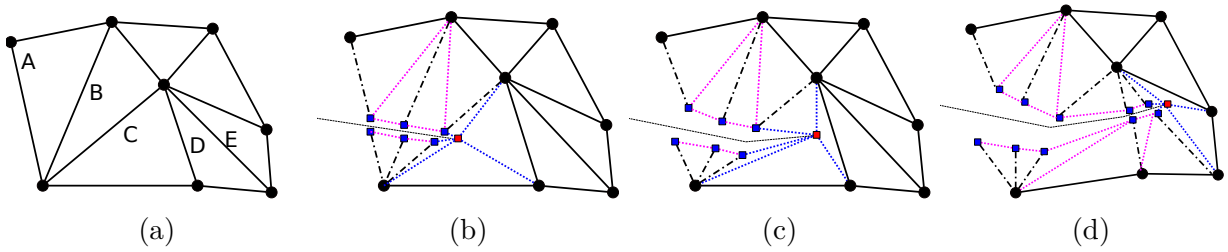


Figure 4.3 The process of cutting the existing surface. a) The initial state of the surface. b) The tool intersected edges A, B and C, generating three new pairs of vertices (blue squares), and stopped inside a triangle, generating a single additional vertex (red square). c) The tool progressed through the surface, but without intersecting any edge and at an angle slightly different from the previous frame; the vertex on the edge of the blade is simply displaced; the model has also been deformed since the previous frame. d) The tool intersected edges D and E, generating two new pairs of vertices and displacing the one on the edge.

For meshing the area that is opened by a cut, we use the concept of a cutting front, similar to Steinemann *et al.* [10]. The novelty in our approach is that the triangles and vertices that are directly on a front are considered *transient* as long as the cut is in progress. The front may be viewed as a moving line that leaves *fixed* vertices and triangles behind as it moves forward through the object. This process of fixing triangles and creating new ones is described in Algorithm 1. During a cutting operation, when a transient triangle becomes large enough – according to a single width parameter w – this triangle becomes permanent and a new set of vertices and triangles is created along the front. On the side of the front, every time the tool edge encounters an intersection, as occurs in Figure 4.3b and 4.3d, it also creates a new triangle along the front. As the tool goes deeper in the model, transient points may be added to or removed from the front as it widens or shrinks. An example of running this algorithm is illustrated in Figure 4.4.

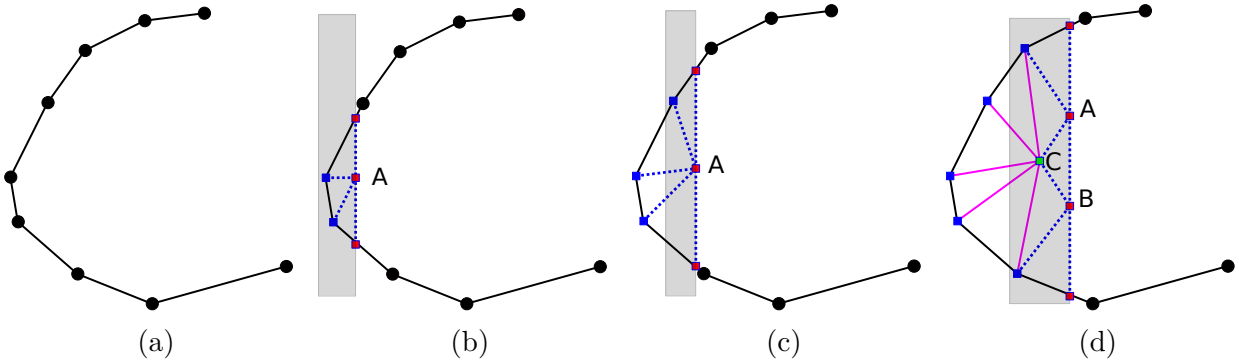


Figure 4.4 Meshing the surface revealed by cutting, using a moving cutting front. Circles represent edges that may be intersected by the cut. The red squares are the vertices composing the cutting front. The gray rectangle is the cutting plane formed by the progression of the cutting tool (from left to right). a) Initial surface. b) Cut starting from the left. The ends of the cutting front are determined by the intersections between the blade and the existing surface. An additional transient point A is placed in the middle of the front. c) The blade progresses, but the generated triangles are still small enough that no new ones are generated, the point A is simply moved forward. d) When moving forward, transient point A was too far from the nearest vertices, so it left a new fixed point C behind; since the front has become too wide, point A has been split in two (generating a new transient point B).

Algorithm 1 Modification of the surface of a simulated object, modeled as a set of triangles, as a cutting edge moves through it.

Input: Existing surface, with a set of transient points where the edge was during the previous frame along with their destination during the current frame.
Output: Surface with new/modified triangles that cover the area swept by the cutting edge during the frame; updated set of transient points.

```
w ← desired max triangle width
for each side transient point s do
    n ← transient neighbor to s
    for each intersection i with a surface triangle do
        i_p ← previous intersection
        Add triangle between i, n and i_p
    end for
end for
for each remaining transient point t do
    p ← nearest fixed vertex
    t_l ← left neighbor of t
    t_r ← right neighbor of t
    Advance t to its final position on the new front
    if distance between t and p > w then
        Place new fixed vertex f at distance w
        Transfer triangles from t to f
        Add triangle with vertices t, f and t_l
        Add triangle with vertices t, f and t_r
        p ← f
        if Front is too wide then
            Split t in two vertices t_l and t_r
            Add triangle with vertices t_l, t_r and p
        else if Front is too narrow then
            Remove t and its two triangles
            Add triangle with vertices t_l, t_r and p
        end if
    end if
end for
```

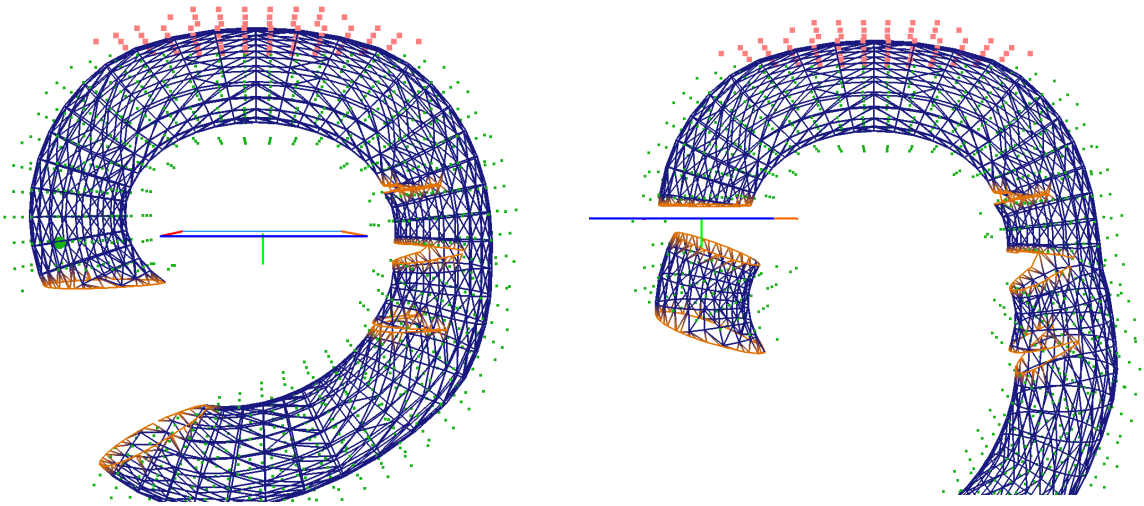
4.3 Results

4.3.1 Versatility and Performance

In order to test the methodology presented above, we implemented the EFG method as well as the described topology definition and modification techniques in the SOFA framework [115].

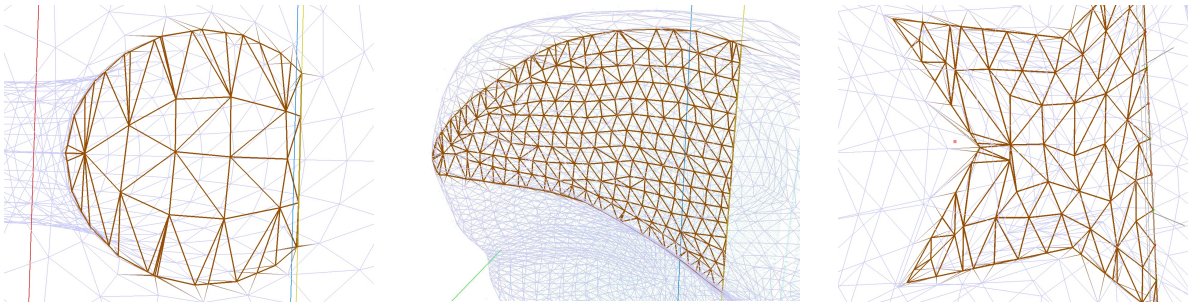
We tested the cutting method in a dynamic simulation to determine how the statically-cut model behaves in real-time settings, when incisions are arbitrarily performed by the user, and whether the topology updates are fast enough to allow for interactive simulation of large objects. The cuts shown in following results were performed within a single frame of the simulation, even when done in multiple steps, since the described method is not yet adapted to an object that is moving and deforming while being cut. This will rather be the focus of future work. The accompanying video showcases the various scenes presented in this section.

Figure 4.5 shows a torus being cut multiple times and with a part that gets fully separated from the rest of the body. No additional degrees of freedom are created along with the cuts, thus resulting in no performance degradation as more cuts are added to the model. The volume topology changes are not fully progressive, but they occur at a resolution slightly lower than half the size of an integration element. As is illustrated in Figure 4.2, the changes happen whenever a link between DOFs and integration points is encountered, so their resolution depends on the number of neighbors (links) around integration points.



(a) (b)

Figure 4.5 a) Multiple cuts were made in a torus. b) A section of the torus was completely separated from the rest of the body.



(a) (b) (c)

Figure 4.6 Detail of the triangles being generated as the cutting tool progresses through the object's initial surface. a) A torus with a coarse surface. b) A liver with a high-resolution surface. c) An object with a star-shaped cross-section.

The algorithm for progressive surface reconstruction generates triangles as those seen in Figure 4.6. The size of the generated triangles is controlled by a single parameter and can thus be proportional to the size of the original surface triangles. Figure 4.6b shows what occurs when the surface has a higher resolution and triangles are much smaller than the width of the cutting front. The new triangles are very regular, except when close to the original surface. In the central area of the newly-generated surface, triangles have a better height-to-width ratio than on average in the entire surface, for all three cases shown in Figure 4.6. In the area that touches the original surface of the object, triangles exactly fit the curvature of that original surface, to the cost of a lower-than-average ratio, meaning these triangles are more stretched than in the rest of the mesh.

Since the method is fully progressive, it does not create additional triangles if the cutting tool only moves by a small amount between frames; instead it only displaces the vertices that coincide with the blade edge. The small number of triangles created every frame makes this method extremely quick, as the results below will show.

The narrow triangles that appear when the ones on the sides of the surface are cut in very small segments are more difficult to avoid and would require a displacement of the initial surface vertices, as in the vertex snapping technique [13]. However, surface triangle quality has no impact on the accuracy or performance of the physical simulation, or on the visual appearance of the model. Lesser triangle quality could affect the performance of acceleration data structures when detecting collisions, but further study is needed to determine whether the benefit is worth the effort of modifying the original surface to merge the smaller triangles.

To determine how the cutting method performs under various parameters and to demonstrate its suitability for real-time simulation, we cut and deformed the same model multiple times, at

different surface mesh sizes and background grid resolutions. The timing for these simulations is shown in Figures 4.7 and 4.8.

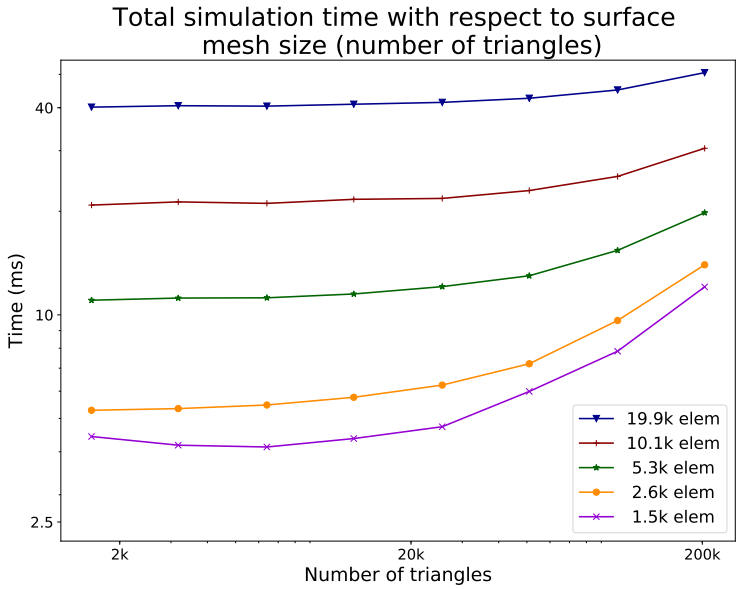


Figure 4.7 Evolution of total simulation time per frame with respect to surface mesh size. Total time consists of deformation, collision detection and cutting time. The different lines correspond to different volumetric model sizes, in number of elements.

Since the cuts are performed before the model is animated, the total frame time consists of the sum of deformation time during animation on the one hand, and collision detection and cutting time during the cutting phase on the other. It should be noted that collisions were detected using a brute-force approach and are thus expected to have relatively less importance in a more complete simulator. Our simulations were run on an Intel Core i5-8400 processor.

From Figure 4.7, we can see that the total time stays reasonable for interactive simulations for volumetric models of up to 10k elements, with at most 30 ms per frame (about 33 updates per second). We can also observe that the influence of surface mesh size (in number of triangles) is more noticeable on smaller volumetric model sizes (in number of elements), with the more pronounced curve of the graph.

These times are comparable to those obtained, for example, by Yeung *et al.* [9] using an element-based method with a pre-factored matrix that is updated as cuts are added to the model. As the cut gets longer, they report update rates dipping to about 50 updates per second for a simple beam model separated along element faces, and to 20 updates per second for a more practical and complex eye model of 15k elements.

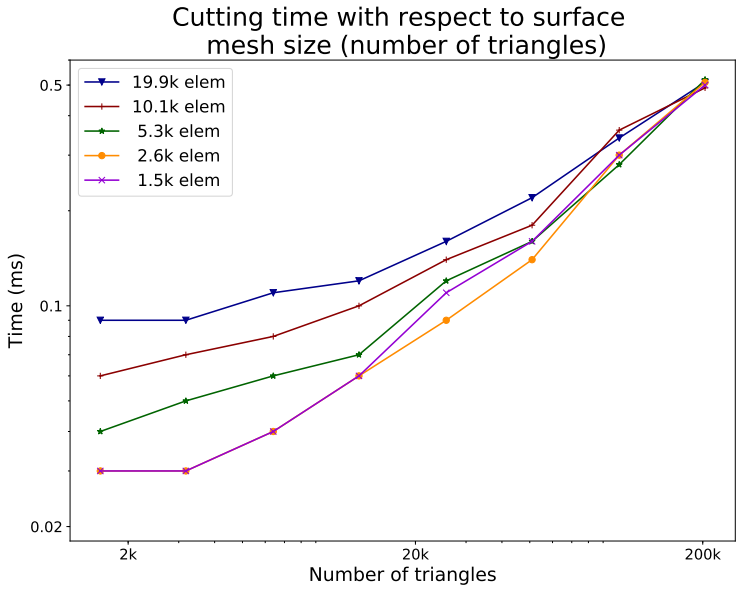


Figure 4.8 Evolution of the time taken to perform all topology change operations, with respect to surface mesh size. The different lines correspond to different volumetric model sizes, in number of elements.

Figure 4.8 shows only the cutting portion of the total time. Cutting time increases with both surface mesh size and volumetric model size, while taking at most 0.5 ms for the combination of the largest for both models (200k triangles and 20k elements). However, surface mesh size seems to have the most weight, since there is little difference between different volumetric resolutions for larger surface model sizes.

The relative importance of the cutting phase is highlighted in Figure 4.9, which plots the ratio of cutting time over total time, in percentage. The clear trends are that increasing surface mesh size increases the cutting time proportion, while increasing volumetric model size actually decreases that proportion. This inverse relationship is to be expected since the number of topological modifications – in both volume and surface models – is proportional to the surface area of the cut rather than the volume of the simulated object. In contrast, deformation calculations directly depend on the number of volumetric elements – or the entire volume, for a given volumetric resolution.

The complete separation of the surface and volume representations also means that their respective characteristics can be determined independently, according to the requirements of the desired application. We can choose to increase the surface resolution to get a more accurate definition and better visuals or the volume resolution if a more accurate deformation is desired. Even a very coarse volumetric object can be cut with an arbitrarily high level of

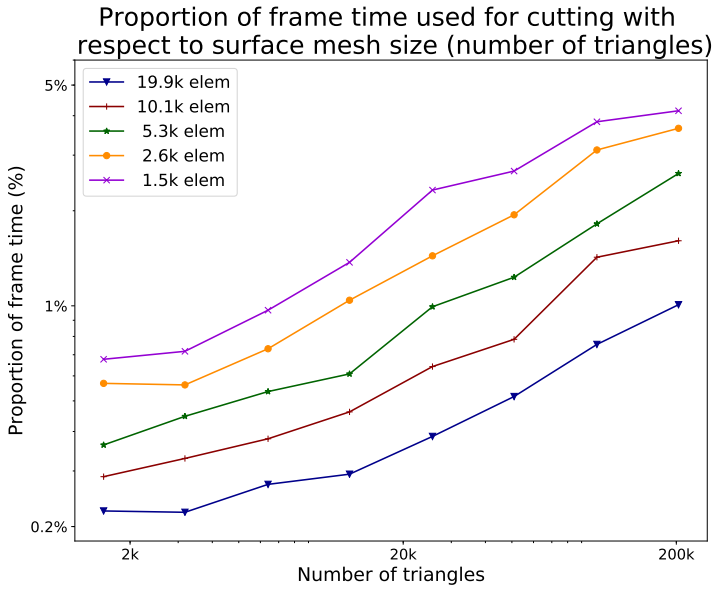


Figure 4.9 Percentage of frame time used for cutting, evolving with respect to the number of triangles in the surface mesh. Individual lines correspond to different resolutions of the volumetric model, in number of integration elements.

visual detail, as shown in Figure 4.10.

4.3.2 Accuracy

Accuracy tests aimed to establish the validity of our cut topology representation as well as that of the algorithm used to select new neighbors for integration elements affected by a cut. To that effect, we compared the deformation of a simple model in our implementation with a finite element method. The finite element model was used as a reference solution in these tests, to verify that the deformation of our method is sufficiently accurate visually to serve in a training simulation, especially after cutting.

Two scenarios were simulated, each involving a hexagonal cylinder, both with and without a transverse cut through half its width in its center, as shown in Figure 4.11. In the first scenario, the beam is fixed horizontally at one end and a downward force (perpendicular to its main axis) is applied uniformly, as gravity would be, bending the beam. In the second one, the force is applied in the same way, but along the main axis of the cylinder, thus stretching the beam. The two scenarios were first simulated with linear finite elements using COMSOL Multiphysics, at a very fine element size (250k DOFs). The scenarios were then simulated using our method at a resolution of about 10k DOFs, which is the number we can reach while still having a real-time simulation, as determined in prior tests. For the tests with the

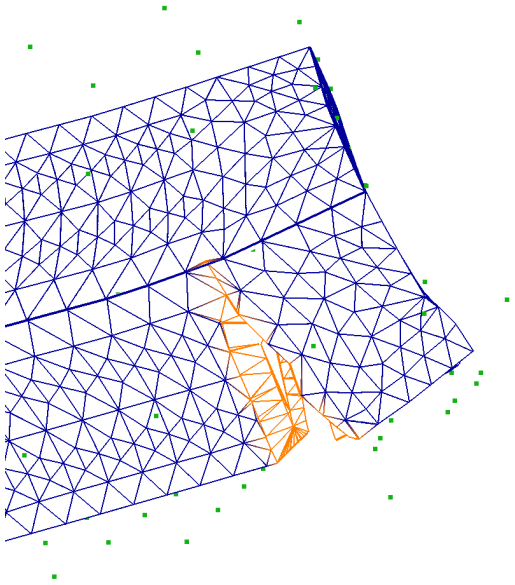
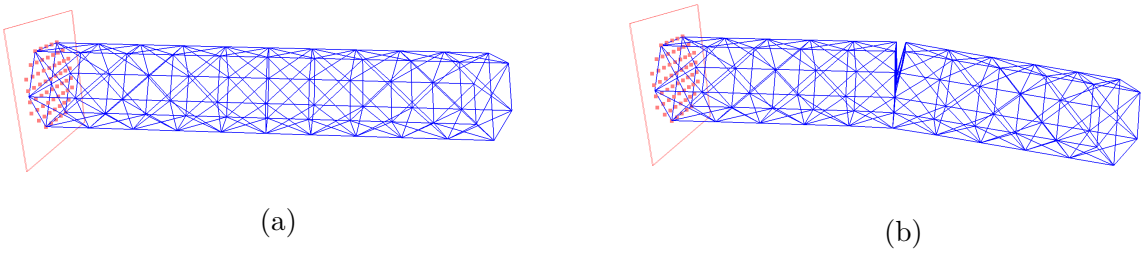


Figure 4.10 A high-resolution surface is mapped to an underlying physical model with lower resolution (DOFs are shown as green dots). The model has been cut and deformed.

cut beam, a cut identical to that of the COMSOL model was created using the algorithm described in section 4.2.3. The average displacement of the tip of the beam was measured, and the size of the error in that measurement was calculated using the result of the FEM simulation as a reference solution. These measurements are presented in Table 4.1.

The results shown in Table 4.1 indicate a very small error, except in the case of the stretched cut cylinder. However, even with a 4% relative error, in a surgery scenario on a 15 cm liver, a displacement of 2 or 3 cm would result in an error of at most 1 mm. In a real-time simulation where the user is receiving both haptic and visual feedback, such an error would be acceptable



(a) (b)

Figure 4.11 Surface model of the cylinder used for testing accuracy. a) Initial model. b) Cut model, after deformation.

Table 4.1 Error in the displacement of the tip of the beam in the tested scenarios with respect to the FEM model used as a reference. The error is expressed in percentage relative to the total displacement of the reference solution (not to the dimension of the model). The two error columns show the result for a simulation with 8 or 4 neighbors per integration point, and one where integration element volumes are not estimated ("One volume").

Scenario	8 nb	4 nb	One volume
Bending (not cut)	0.4	0.7	7.7
Stretching (not cut)	0.6	1.4	3.2
Bending (cut)	0.8	4.2	9.7
Stretching (cut)	3.7	23.1	18.1

as long as the perceived force remain consistent with the visualized displacement.

To verify the usefulness of estimating the volume of material inside each integration element, we repeated the simulations without that estimation. The results appear in the final column of Table 4.1. Combined with the results from the 4- and 8-neighbor simulations, this suggests that both estimating the volume and smoothing the distribution of material among DOFs are important factors in carrying out an accurate simulation.

4.3.3 Limitations

Volume cutting only progresses as links between DOFs and integration points are severed. While this is at a finer resolution than an integration element size, it still occurs in discrete increments. Furthermore, the mass is defined at integration points before being distributed to neighboring DOFs, but it is not redistributed between integration points when a cut is made within an element. As a result, DOF masses on either side of a cut may not be properly assigned.

Multiple cuts that intersect each other do not pose any particular problem to this method. However, when they occur within the same integration element, that element may quickly become too isolated from the rest of the object and its displacement can no longer be computed.

The visibility criterion used for splitting the topology means that when the object is highly deformed and curved, the blade might either intersect or miss links that it should not. This would mostly be a problem when the linking DOFs and integration points, representing that visibility criterion, move away from the visual representation of the object's surface, either towards the inside or the outside.

As for the surface generation algorithm, a very simple heuristic is used to determine where

and when new vertices and triangles are placed, which may produce irregular triangles in some cases. While the meshes on which we have tested this algorithm all produced triangles that were more regular than on average, there are no firm guarantees on triangle quality.

4.4 Conclusion and Future Work

We presented two new methods: one for representing and quickly modifying the topology of a deformable object simulated with an EFG method, and an algorithm to progressively update a mesh-based surface representation of the same object. With these methods, topology changes may be introduced in a model at real-time rates with a very low impact on frame computation time. Cuts added to the physical representation of the object do not introduce any new degree of freedom and subsequent deformations have a low error with respect to the relative displacement of material points. Cuts added to the surface representation result in a low number of triangles on the newly created surface, even with very small movements of the cutting tool.

Future work will focus on developing a more fully interactive way to simulate the cuts while they are being created, using physical criteria rather than simply the position of the tool on the object, and adding a capability for consistent force feedback as the surface and volume are modified. We will also investigate the possibility of redistributing the volume when cuts occur in the middle of integration elements to get a more accurate deformation in these cases.

Acknowledgement

This work is funded by the Natural Sciences and Engineering Research Council (NSERC) under grant 501444-16.

CHAPITRE 5 DYNAMIC CUTTING OF A MESHLESS MODEL FOR INTERACTIVE SURGERY SIMULATION

Authors: Vincent Magnoux and Benoît Ozell

Presented at *Salento AVR 2020*, September 2020

Paper number: 3043

Abstract

Virtual reality has become a viable tool for training surgeons for specific operations. In order to be useful, such a simulation need to be as realistic as possible so that a user can believe what they experience and act upon the virtual objects. We focus on simulating surgical operations that require cutting virtual organs. They offer a particular set of challenges with respect to simulation stability, performance, robustness and immersion.

We propose to use a fully continuous movement representation and collision detection scheme between cutting tool and other simulated objects to improve the robustness of the simulation and avoid breaking immersion with errors in topology changes. We also describe a new way to attach the surface of a simulated object to its underlying physical model, consistently while it is being cut. This feature helps maintain immersion by keeping the visual aspect of the object coherent with its physical behavior and by allowing correct transmission of actions from the user on the object.

Our tests show that the proposed tool movement representation properly generates continuous cuts in simulated models, even as they move and deform. It also allows cutting when a moving model comes into contact with a motionless tool, and to model a curved or deforming tool without additional effort. Our surface mapping method results in a visual model that closely follows the movement and deformation of the physical model after it has been cut.

5.1 Introduction

Surgery simulation technology using virtual reality has become a useful and practical tool for learning specific operations like lung tumor removal [137], aneurysm clipping [138] and many others [139]. As with all virtual environments, an important aim for these simulations is to create a world in which the user can feel immersion and presence as much as possible. This sensation may come from how the world is perceived by the user's senses, such as

vision, hearing and touch, and from how the world reacts to the user's actions. The research presented in this paper focuses on the latter aspect, more specifically on how virtual objects like organs are modified through cutting actions from the user during a surgical operation.

There has been a lot of progress in cutting simulation, both on the physical and the visual aspects, yet there remain open challenges to improve the realism of this interaction. Our goals are to develop a tool representation that increases the *robustness* of the simulation during a cutting action and to maintain *coherence* between the visual surface and the underlying physical model at the same time. Together, these objectives will improve simulation interactivity by providing a more accurate representation of hand movement, avoiding breaks in immersion in a straightforward way and ensuring a realistic reaction of soft tissue to user action.

On the one hand, we improve the robustness by ensuring that any part of a simulated object traversed by a cutting tool will indeed be cut. This means that no geometric element will be missed due to the discrete nature of the simulation or to the movement and deformation of either the object or cutting tool.

On the other hand, we preserve coherence between the visual and physical models by elaborating a set of criteria that determine how the surface mesh moves with the physical model and how it can transmit forces to it.

5.1.1 Background and Related Work

Soft body simulations that allow cutting may be classified according to the physical simulation method, which may either be mesh-based or meshless. A thorough review of the various cutting methods may be found in [140]. Mesh-based cutting involves removing and recreating elements along the tool trajectory [14, 21, 42, 43, 135, 136], while meshless methods rely on updating the connection or visibility between the particles that form the physical object [68, 69,73,87,125]. For both simulation categories, cutting requires modeling the trajectory of the tool as well as finding the intersection of the tool with the object along that trajectory. At the lowest level, this intersection is often computed as the contact between edges or between triangles and points of either model.

Tool Representation.

Most simulations represent the cutting part of the tool as a one-dimensional edge composed either of a single segment [19,24,47] or multiple ones [67,136]. The area between the position of the cutting edge during the previous frame and its position during the current frame

is considered the *swept area* [67]. It may be approximated by a plane [19, 24, 47] or be triangulated to obtain a more closely fitting cutting surface [27, 28, 65, 69, 73].

One major drawback of this approximation is that it does not capture well the movement of the cutting edge when it does not lie in the same plane between two frames. This may cause some contact detection to be missed, eventually resulting in erroneous modifications in the modeled object surface. [24] propose to use a continuous representation of the cutting edge, which partially offsets this problem by providing a more realistic approximation of the edge movement. However, it still fails to take into account the movement and deformation of the object being cut.

There are however approaches that go in a different direction. A single point with an orientation may be used to represent the tool [75, 123]. Together, the position and orientation of the point define a plane which can be used to compute level sets at every frame, which in turn determine where different material points are with respect to the tool. In other cases, some represent the tool with a volume and check for an instantaneous (discrete) intersection between the tool volume and the object volume [135, 141]. In a similar way, [21] uses as a cutting surface the intersection of the tool volume with the object, represented with voxels – resulting in an approximate cut. Finally, rather than using a well-defined geometric shape, the tool volume may simply be sampled with a set of points [68].

Surface Mapping.

In general, simulations may either use an implicit or explicit surface representation to draw an object on the screen. However, implicit representations usually require a lot of computing power and are thus not well suited for an interactive simulation [27, 57]. Some manage to perform volume- and point-based rendering in real time on parts of a model using surface splatting [125, 126] or, more recently, ray casting on the entire object [123].

Explicit surface representations must be somehow attached, or *mapped*, to the physical object model so that they can properly move according to the results of the physical simulation and continue to transmit interaction forces.

For mesh-based physical models, the trivial case consists in using the boundary faces of volume elements to display the surface [14]. However, the resolution of the surface is thus directly tied to that of the physical model. A separate representation allows to display the object in much greater detail than it is possible to physically simulate it.

With mesh-based physical models, this mapping between the surface mesh and physical object is done by selecting to which volume element a surface vertex belongs and computing

its barycentric or trilinear coordinates within that element [20,21,42,43,135,136,142]. When the element is displaced or deformed, the new vertex position is computed using the mapping coordinates in the new element configuration.

With meshless models, the mapping is done by selecting a set of appropriate particles and computing a set of weights for each of them relative to the vertex. Instead of weights calculated through barycentric or trilinear coordinates – since there is no geometrically set element – a moving least squares (MLS) scheme is often used [27,68,69,73,125,143]. Other weighted mappings are also possible, based on how the displacement field is sampled in the physical simulation [87,144,145].

5.1.2 Contributions

This paper describes the following contributions:

— A fully continuous cutting interaction between a tool and a surface that includes detecting the cut primitives and modifying the surface mesh in consequence. Both the tool and the object may be deforming during that interaction.

— A small set of criteria that allow mapping the surface on its underlying volume model to maintain visual coherence.

Section 5.2 describes the details of these contributions, while Sect. 5.3 demonstrates their utility through a set of examples.

5.2 Method

5.2.1 Tool Representation and Collision Detection

For the purpose of cutting, the interacting part of the tool is represented by a series of points joined by straight line segments, collectively called the *cutting edge*. The only interaction on which we will focus in this paper is for cutting, without exchanging forces between the tool and other simulated objects. The cutting edge is considered ideal, in the sense that it has no volume, cuts as soon as it comes in contact with a simulated object and does not generate any force on it. In addition to the cutting tool, the term *edge* will refer to the primitives used for detecting intersections. For the surface, they are the lines between surface vertices that form triangles and for the underlying physical object, they are the links between integration points and particles that form its topology.

The object itself is deformed and cut using the method described in [72] and [146]. For the purpose of topology modifications, the object is entirely composed of edges, as defined above.

Cutting only occurs when edges are intersected, not when a point enters a triangle on the surface. Furthermore, we only mention cutting the object's surface, but everything discussed in this section applies in an identical way to cutting the edges that define its volume.

As the tool is displaced from one frame to the next, each of its edges form a skew quadrilateral with the four points being the two ends of the edge at the first frame and the two ends at the second frame, as shown in Fig. 5.1. The set of 3-dimensional quads form the surface swept by the blade during the frame. When detecting collisions, the deformed object's surface is similarly only considered as a set of edges that move through space between animation frames. The detection is made between each pair of edges that belong to the separate objects, according to the continuous collision algorithm described in [147].

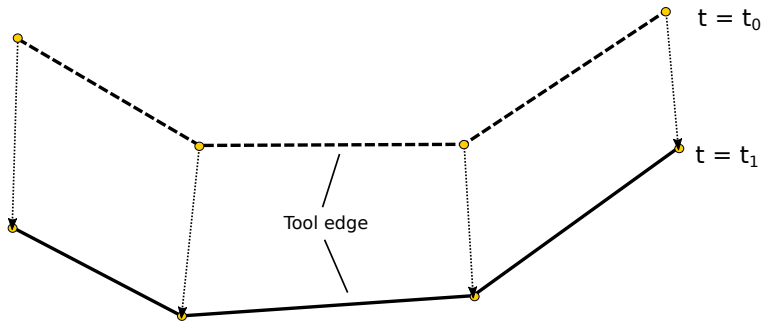


Figure 5.1 The area swept by the cutting tool is only described in terms of the position of its edge at the previous frame (t_0) and its position at the current frame (t_1).

We assume each point of the blade moves in a straight line during a frame. This approximation remains accurate as long as the displacement between frames is relatively short, which is the case when animating the simulation at a rate of 60 frames per second, leaving less than 17 milliseconds per frame.

This operation is more complex than with an explicitly-triangulated swept surface since it requires continuous edge-edge collision tests rather than just a series of discrete triangle-edge tests. However, there are several situations where it provides a correct solution, in contrast to a discrete detection scheme.

Situation 1. The continuous method allows for a completely gap-less detection of cut edges, which is especially useful when both objects are moving between frames. For example, an edge from the deformed object could move across an entire swept area quadrilateral during a frame time, leaving it undetected when using a discrete scheme. Figure 5.2 illustrates such a situation in two dimensions. The cutting tool is displayed as a wedge in this diagram, but only the tip has any physical presence in a 2D setting. Its movement traces a line – rather

than a swept area in 3D – which can be used for detecting intersections with surface triangles.

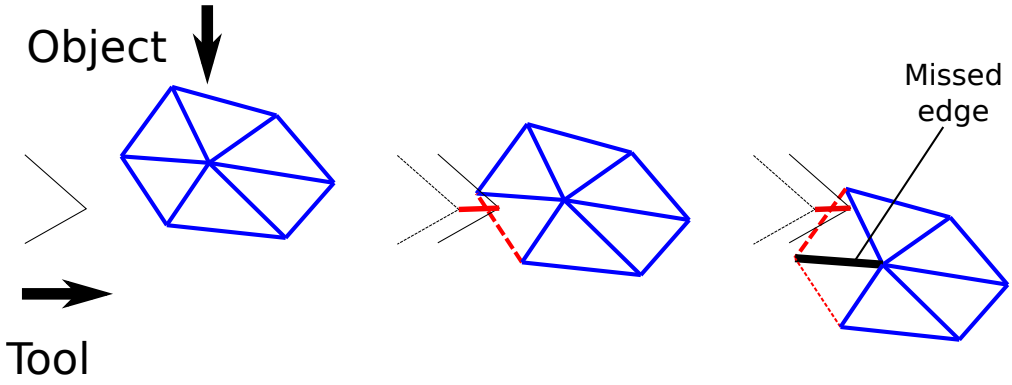


Figure 5.2 Example of *Situation 1*, in 2D, where a collision detection scheme that is continuous only with the tool movement would fail to detect a certain triangle edge. The large arrows indicate the direction of movement of the tool and object. The red line between tool positions represents the space swept by it between frames.

Situation 2. That method also avoids detecting cuts in places where a surface edge enters the swept area *after* the blade has passed, as in Fig. 5.3. In that case, the tip of the cutting tool was always outside the object, but the line that connects the current position to the previous one still intersects some edges in the object at its current position.

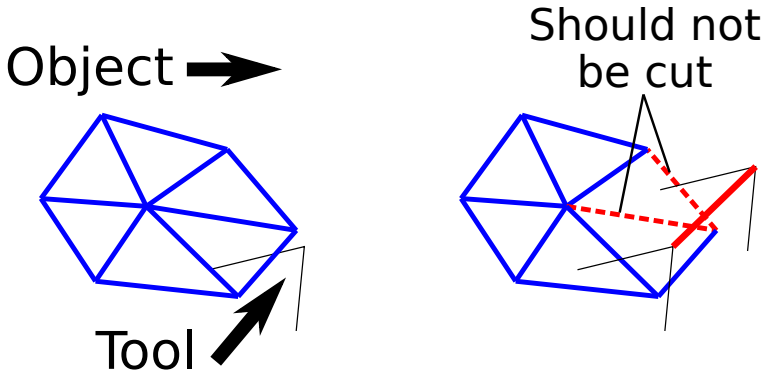


Figure 5.3 Example of *Situation 2*, where the line spanned by the tip of the cutting tool intersects two of the objects edges at the end of the frame, even though the tool was outside the object at every point in time between the two frames. The large arrows indicate the direction of movement of the tool and object. The red line between tool positions represents the space swept by it between frames.

Situation 3. Another case where this description is essential is when the blade is not moving but the deforming object is, whether under gravity or elastic forces. In such a case,

the blade does not sweep any area, but the object must still be cut, as shown in Fig. 5.4.

When the tool is moving, the continuous method also provides a slightly more accurate approximation of the swept area as well as a more accurate point of contact, both in space and in time, since it is not flattened by using triangles.

Since the tool may also cut while not moving, we must take particular care with defining some quantities. For example, we might want to know on which side of the cutting edge lie each pair of vertices. In our method, the new normal on the surface is entirely determined by the position of each intersection – the trajectory of the cutting tool relative to the object. This avoids relying on the normal of a swept surface that does not exist when the tool is motionless.

One point worth mentioning is that with the movement representation described above, the cutting tool itself can be deformed during the simulation without the need to handle that case differently.

5.2.2 Surface Mapping and Remapping

To ensure that the surface remain coherent with the movement of the physical simulated object and thus provide a proper interface with other virtual objects and with the user, we establish a set of constraints that determine how each surface vertex moves with the physical object model. These criteria are tailored to the deformation and cutting method of [146], where the particles are embedded in a regular background integration grid; however, they can be applied almost directly to any method that maps a triangulated surface on a set of particles.

Since the displacement of a material point inside the object is determined by interpolating the displacement of neighboring particles using shape functions computed with an MLS scheme,

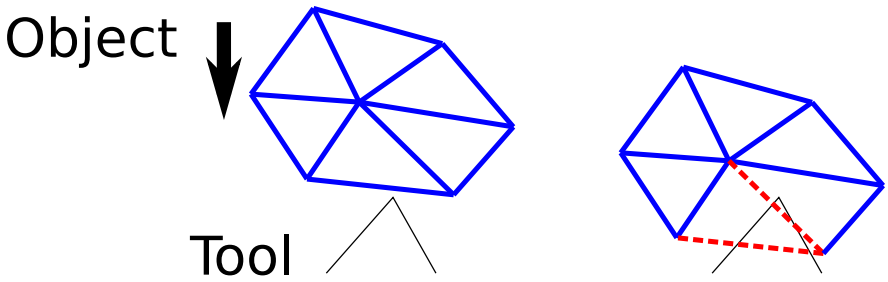


Figure 5.4 Example of *Situation 3*, where a simulated object moves downward onto a motionless cutting tool. The tool's edge does not sweep any area, but must still be able to cut the object.

a natural solution is to use the same scheme for surface vertices. The set of particles that determine the position of a point will be called the *mapping* of that point in the remainder of this paper.

The main challenge with this method consists in choosing the right particles on which to map a vertex. This choice does not simply depends on the distance between the vertex and particles, because we need to take into account the initial topology of the object as well as the cuts introduced with the tool. We may however choose that criterion as a starting point for the set of constraints, since it is how it would be done for a convex object that does not contain any cut.

It should be mentioned that when determining where to map a given vertex, only the rest configuration of the object is considered. While intersections are detected in the current (deformed) configuration, their location in the object at rest can be determined in a straightforward way: since they always occur on edges, between two points, they can be fully described by a proportion along the vector connecting the two points. For simplicity, we consider that proportion to always be the same, whether in the rest or deformed configuration.

To avoid mapping a vertex to particles that belong to physically separate parts of the object – while still being geometrically close – we first decide to select particles that are in the neighborhood of a single integration point. This guarantees that they will all move in a locally coherent way. The mapping problem is thus reduced to finding a single most suitable integration point for each vertex of the surface mesh. This approach is sound also because the integration elements are more closely related to the geometry of the object – they distribute its volume and mass to the particles. They are all at least partially inside the object, whereas particles may lie slightly outside the surface of the object as described in [72].

When the object contains a cut, or simply a concave portion, the integration point nearest to a certain vertex may actually be in another part of the object (see Fig. 5.5). To avoid any such wrong mapping, we add as a criterion that the integration point on which a vertex is mapped must lie below the plane defined by the vertex and its normal.

There are however many cases where the plane test by itself is not sufficient to determine the right mapping point. For example, in Fig. 5.6, when the cutting tool passes between a certain vertex and the integration point on which it is mapped, using only the criteria mentioned earlier, we would try to map the vertex to the same integration point. We therefore associate with the vertex an additional plane below which the mapped integration point must also be, whenever a new cut is introduced into the object near that vertex.

The final criterion we need to define allows to take *all* introduced cuts into account – rather

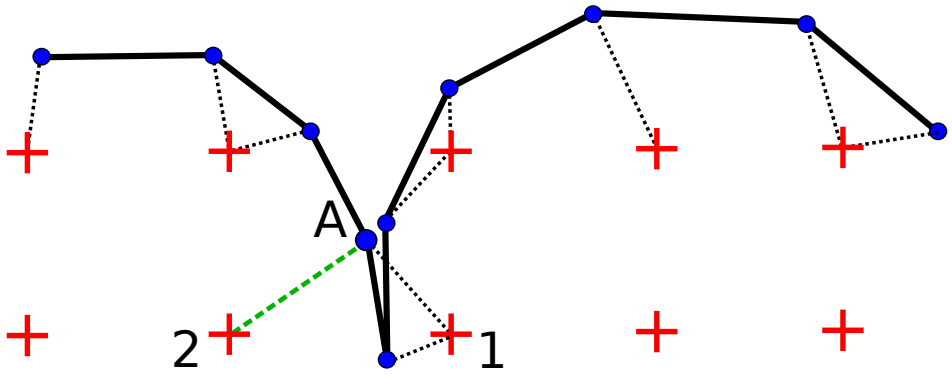


Figure 5.5 Illustration in 2D of a situation where the nearest integration point to a certain vertex would not be the best mapping. Integration points are displayed as red crosses and vertices as blue circles, with dotted lines indicating the nearest integration point. Vertex A has integration point 1 as its closest, which lies across another surface boundary. We would rather choose an integration point that is *not* separated by another part of the surface, such as point 2, connected to the thicker green line.

than just the most recent one – without specifying an arbitrarily large number of cutting planes. For each vertex, we only choose as potential mapping targets integration points which are connected to a set of particles *similar* to that of neighboring vertices. In other words, we need to consider not just the integration point, but the set of particles on which vertices are mapped. For an integration point to be considered valid, it has to have at least one particle in common with every vertex that form a triangle with the unmapped vertex. This also ensures that vertices that are close to each other will be mapped to integration points that are also close to each other, keeping the displacement of the surface locally coherent.

In summary, when choosing a new integration point to which a vertex will be mapped, that point must be

— connected to some of the same particles on which neighboring vertices are already mapped;

— below the plane tangent to the surface at that vertex;

— below an additional plane defined when a cut was introduced.

From the integration points that match all these criteria, the one closest to the vertex under consideration is chosen.

5.2.3 Dynamic Simulation

Whenever a part of the object is intersected by the cutting tool, all affected vertices must find a new mapping. We define in the next paragraphs which surface vertices to remap after

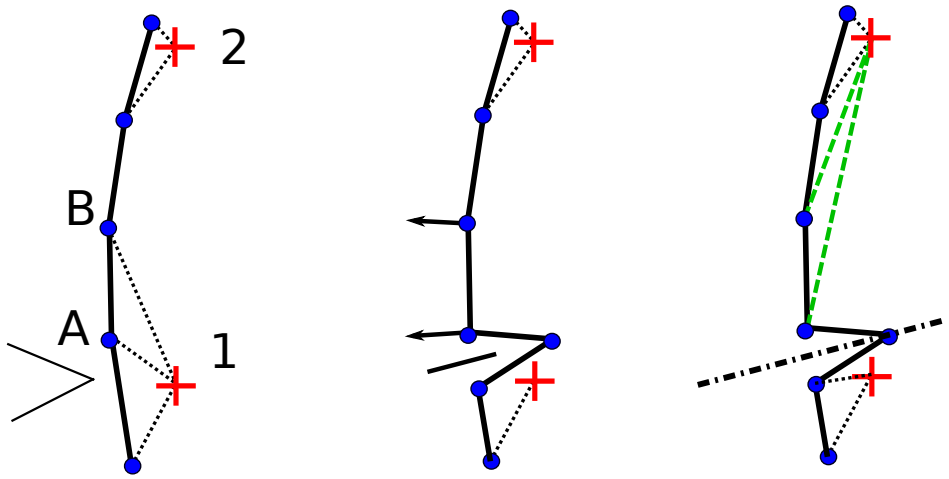


Figure 5.6 Illustration of the need for an additional constraint plane when finding a good mapping. *Left:* A cutting tool approaches from the left, below vertex A. *Middle:* The introduced cut has caused vertices A and B to lose their previous mapping. Their initial normal is indicated. Using only that normal would cause them to be mapped on the integration point 1, from which they were just unmapped. *Right:* The introduced cutting plane causes vertices A and B to be mapped to point 2.

an interaction with the cutting tool.

When a surface edge is cut, two vertices are added: they must be mapped. The endpoints of the (now cut) edge must be remapped, and receive an additional plane corresponding to the surface normal of the cut (see Fig. 5.6, right). There is also the case where the blade intersects the segment between a vertex and the integration point on which it is mapped. In that case, the vertex has to be remapped and be associated with an additional plane. That new plane passes through the cut point and has the vertex-integration point line as a normal, as if that line were just another surface edge.

Adjustments to the mapping also need to be made when the underlying physical model is cut. A modified connectivity between particles and integration points changes the eligibility of affected integration points in certain cases, so any vertex that is mapped to these points needs to be remapped. For example, two integration points that share a certain particle may become separated and no longer be considered as neighbors, affecting how nearby vertices must be mapped.

When the mapping of a vertex changes, its position at the next frame may change slightly, even if the object is not moving. Using a fully continuous collision detection scheme prevents any missed surface edge.

Animation Loop.

The order in which the main steps of the simulation are carried out is important to obtain a consistent interaction between the dynamic tool and the object. An animation step in our current implementation consists of four operations:

1. Process user input. This only consists in moving the cutting tool to its new position, as determined by a haptic device, for example.
2. Detect collisions. These collisions are currently only for cutting.
3. Perform topology changes, based on the intersections detected in the previous step. This results in updated data structures describing the object – physical model, surface and mapping between the two.
4. Solve the dynamic system. This system incorporates internal elastic forces, gravity and external forces. This step also updates the surface vertex and integration point positions according to the new particle positions given by the system solution, as described in [72, 146].

In such a scenario, input by the user is considered to occur during the displacement of surface positions, even if it is only processed after. Any movement from the user that happens during the animation step will have an effect based on the *next* surface and object position.

5.3 Results

The applicability of our method is demonstrated through an implementation using the SOFA framework [115]. We show examples of various situations described in the previous section where it is useful, both on simple geometries and on organ models. The proper functioning of the surface mapping criteria is apparent is some of these scenes, but is most visible in the accompanying video.

The most obvious advantage of having a fully continuous collision detection scheme, is that we can still properly detect tool-object intersections when the blade is static and the object is moving. This can occur for example under the action of gravity or elastic forces. This capability contrasts with the usual method where only the cutting edge is considered to be moving from one frame to the next. Figure 5.7 shows three different frames of a simulation where a circular cylinder is falling on a simple straight cutting edge. Even if the edge does not sweep any area, the object can still be cut. This example also shows how the surface position remains consistent with the position of the object. Surface vertex positions are entirely dependent on the position of the underlying physical model, not shown in this picture.

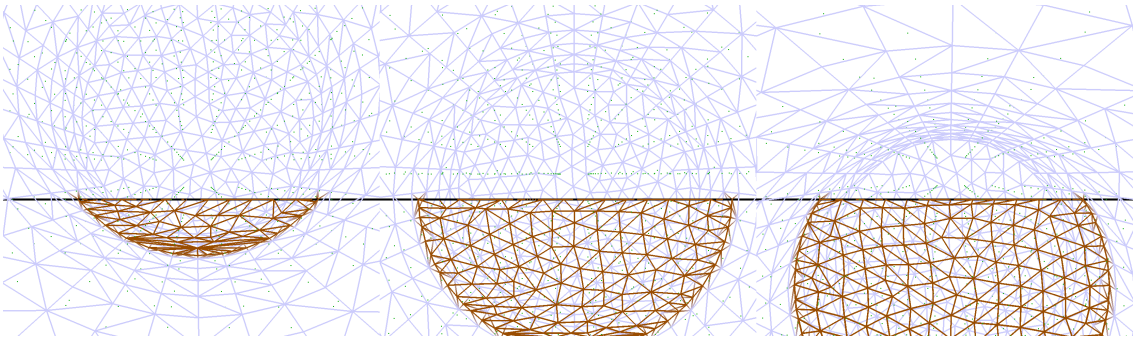


Figure 5.7 A circular cylinder is moving downward and being cut by a horizontal edge that remains in the same place. The surface of the cylinder (including the newly-generated part) properly follows the movement of the underlying physical model.

As a comparison, Figure 5.8 shows a similar scene, this time where the circular cylinder is static and the cutting edge is moving upward. One can see that the resulting cut surface is very similar to that of Figure 5.7. The surface generation method remains identical; the only difference is the sequence and speed at which triangle edges are intersected.

Another major advantage of the fully continuous motion modeling is that the cutting edge will never miss a triangle edge because of their movement between frames – barring any numerical error. This can occur in many situations, especially when the blade movement is almost parallel to a surface edge, or when it undergoes a twisting motion relative to the surface. Figure 5.9 displays such a situation, where a curved blade is rotating while advancing and cuts a surface edge that is almost parallel to its local motion.

In a surgery simulation, most simulated objects are more complex than those shown so far. Our proposed tool representation and surface mapping methodology are not affected by the complexity of the model being cut, in theory. The difficulty lies mostly in how to manage the cuts that are detected. Once the surface and physical model are cut, the mapping can

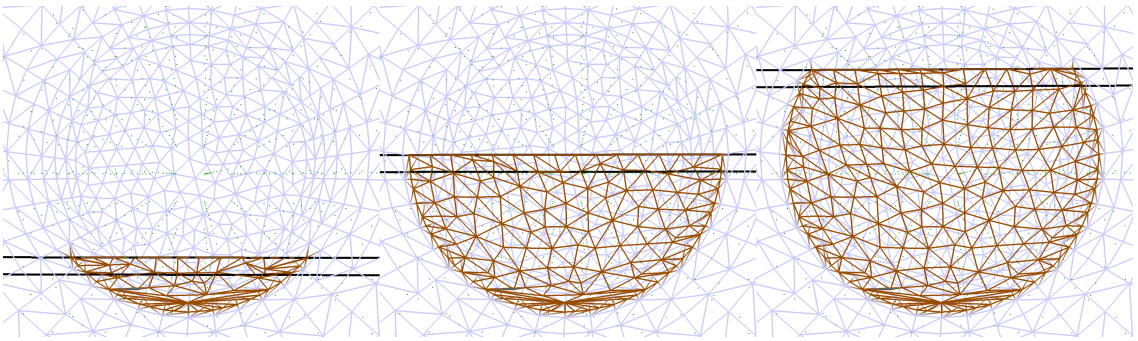


Figure 5.8 A horizontal cutting edge moves upward in an fixed circular cylinder and cuts it.

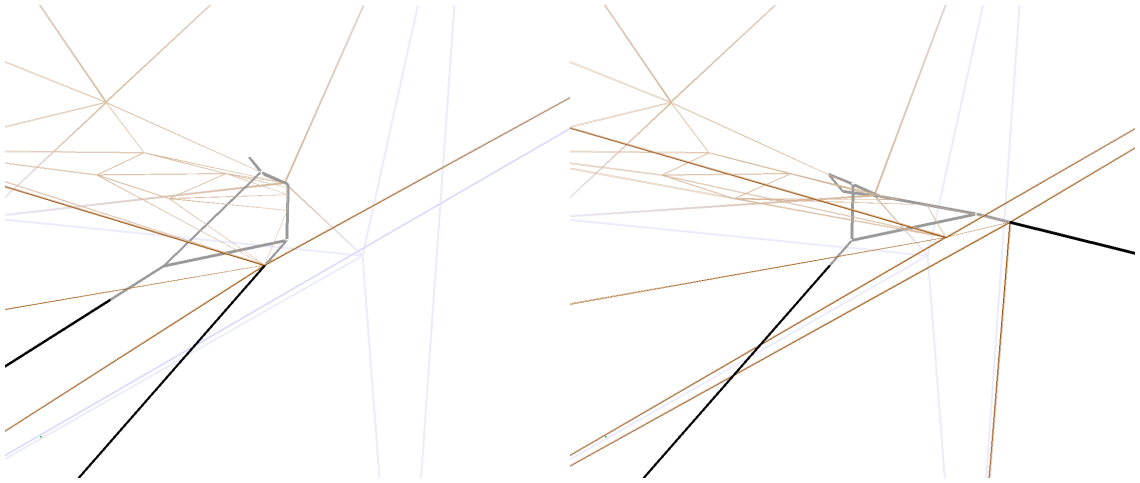


Figure 5.9 Cutting edge (in black) that is curved and that twists as it moves forward.

proceed as with any other model. Figure 5.10 presents a brain hemisphere being cut by a slightly curved blade. This model has many grooves and folds that the blade enters and leaves simultaneously or at different times. This gives rise to situations where the cutting edge traverses several separate parts of the model in a given frame. Our method handles these multiple fronts merging and splitting as the blade moves forward the same way as it does a single front.

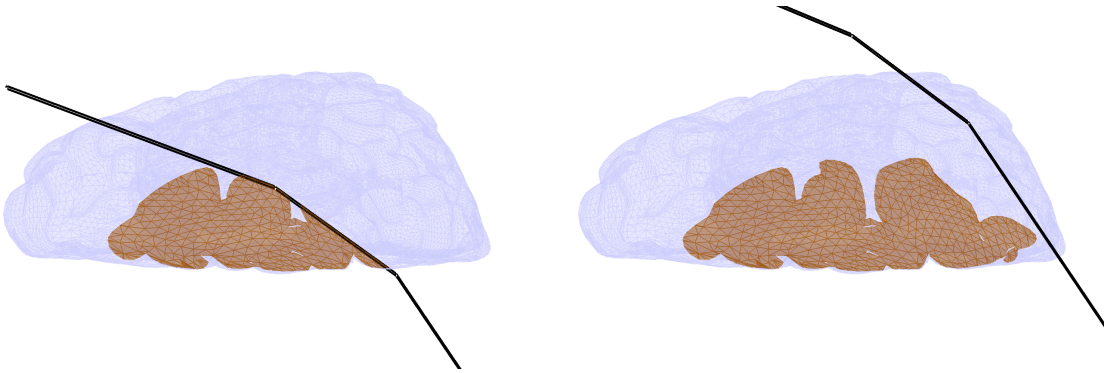


Figure 5.10 Cutting a more complex model that may appear in a surgical simulation scenario. The cutting tool is able to deal with the numerous folds present in a human brain.

Figure 5.11 shows that a single object may be cut multiple times with the same tool. In the first image, the cut surface appears to be slightly curved, because the scene was animated during the cutting operation and the liver was slowly tilting down as the blade went through it. Figure 5.11 also shows that the displacement of the surface remains consistent with the cuts introduced by the tool.

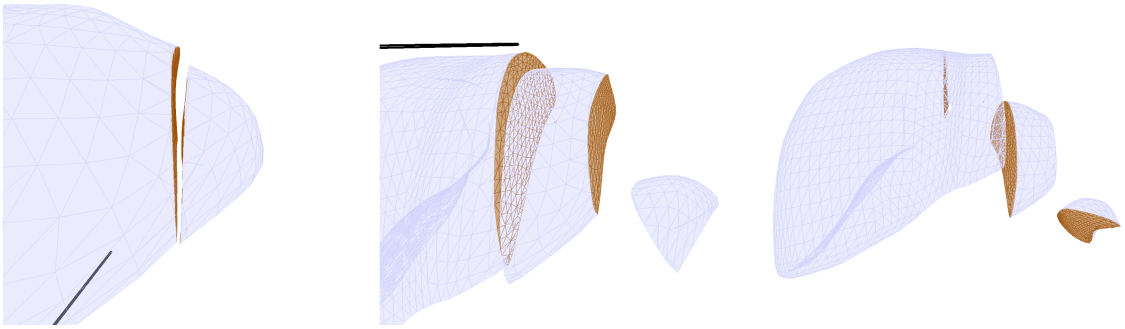


Figure 5.11 Illustration of a series of cuts in the same object. A slight curve is noticeable in the first cut and is caused by the movement of the liver under the effect of gravity while the cut was performed.

Figure 5.12 offers a more detailed perspective on how surface vertices are mapped on the underlying physical model. On the cut surface, it shows that each mapped vertex is linked to an integration point on its own side of the cut. That point is generally the closest that fits all mapping criteria, taking the new cut into account.

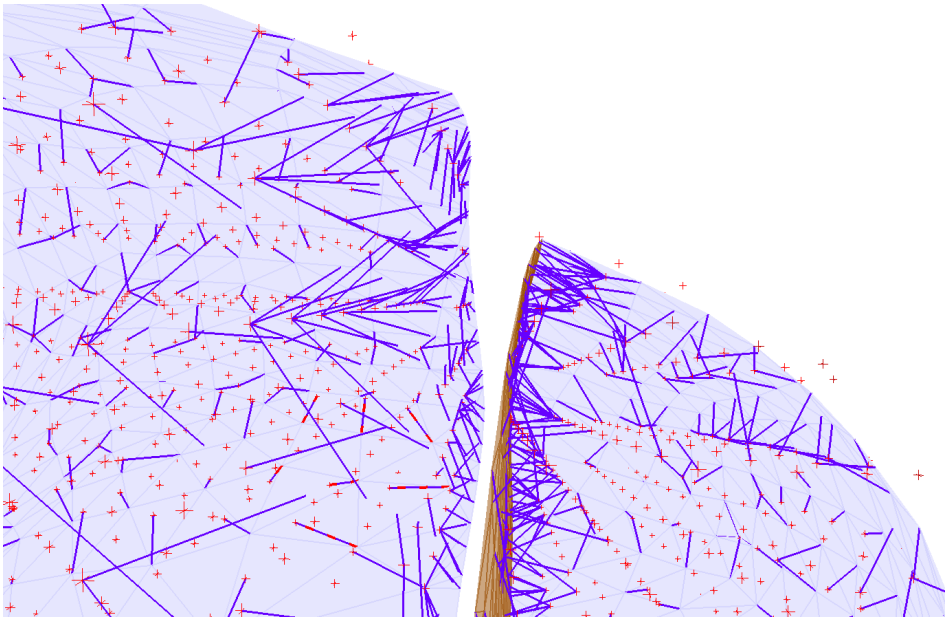


Figure 5.12 A view of the liver cut scenario that displays the integration points (red crosses) and highlights to which point each vertex is mapped (blue lines). Each vertex is properly mapped to a point on its own side of the cut.

Limitations.

The correctness of mapping a surface based on the criteria described in Section 5.2 depends greatly on the relative size of the surface mesh and that of the physical model on which it is mapped. The surface mesh *must* be at least as dense as the integration point grid. Otherwise, it will not be possible to find a point near a vertex that is somewhat close to every triangle formed by that vertex.

A related issue is that of the smallest possible cut in a model. While it is possible to create and display a surface with an arbitrary level of detail, it is not possible to cut these details away from the rest of the simulated object if there are not enough integration elements under them. If a sufficiently small part of the surface is separated from the rest of the object, it will not be possible to map it to the underlying physical model. With the scheme used for this research for distributing particles and integration points, the thinnest part of an object that can be fully separated from the rest and still be properly mapped is about the width of two integration elements.

5.4 Conclusion and Future Work

We have described and demonstrated the use of a dynamic tool representation that allows to interactively cut a deforming object in a variety of situations: with a static, moving or deforming tool on a static, moving or deforming surface. We have also described a method to successfully map the cut surface of the object on its underlying cut physical model, itself based on a set of particles and a background mesh.

Together, these two methods lead to a more robust and coherent simulation, improving the immersion and interactivity of the virtual environment in a surgical setting. This work forms a component that will become part of an existing simulation system.

That system displays a surgical scene on a 3D screen, offering sufficient visual immersion for accomplishing tasks on the relatively small work area of the surgery site. It is also equipped with a haptic device that provides a virtual surgical tool as the single means through which the user can interact with the scene. Using that system will enable us to make our method available for testing by surgeons and will be the subject of future research.

Acknowledgement

This work is funded by the Natural Sciences and Engineering Research Council (NSERC) under grant 501444-16, in collaboration with OSSimTech.

CHAPITRE 6 GPU-FRIENDLY DATA STRUCTURES FOR REAL TIME SIMULATION

Authors: Vincent Magnoux and Benoît Ozell

Submitted to *Simulation Modelling Practice and Theory* , July 2020.

Reference number: SIMPAT-D-20–852

Abstract

Simulators for virtual surgery training need to perform complex calculations very quickly to provide realistic haptic and visual interactions with a user. The complexity is further increased by the addition of cuts to virtual organs, such as would be needed for performing tumor resection. A common method for achieving large performance improvements is to make use of the graphics hardware (GPU) available on most general-use computers. Programming GPUs requires data structures that are more rigid than on conventional processors (CPU), making that data more difficult to update.

We propose a new method for structuring graph data, which is commonly used for physically based simulation of soft tissue during surgery, and deformable objects in general. Our method aligns all nodes of the graph in memory, independently from the number of edges they contain, allowing for local modifications that do not affect the rest of the structure. Our method also groups memory transfers so as to avoid updating the entire graph every time a small cut is introduced in a simulated organ.

We implemented our data structure as part of a simulator based on a meshless method. Our tests show that the new GPU implementation, making use of the new graph structure, achieves a 10 times improvement in computation times compared to the previous CPU implementation. The grouping of data transfers into batches allows for a 80-90% reduction in the amount of data transferred for each graph update, but accounts only for a small improvement in performance. The data structure itself is simple to implement and allows simulating increasingly complex models that can be cut at interactive rates.

6.1 Introduction

Despite decades of progress, realistic real-time surgery simulation remains computationally challenging. Calculating the deformation and behavior of organs according to physical models

is complex and needs to be done very fast to produce a virtual environment that is responsive to user actions, especially when haptic feedback is desired. The calculations become even more demanding when cutting operations need to be performed, such as the resection of a tumor, while the organ and surrounding tissue are being deformed. The behavior of soft tissue is itself non-trivial to simulate, but adding cuts and other topology-changing operations means that many acceleration structures that allow increased performance can no longer be precomputed.

One approach to improve the resolution and realism of simulations consists in making better use of the constantly increasing capacity of common computer hardware, such as multi-core processors (CPU) and graphics processors (GPU).

In general, multi-threaded CPUs allow performing multiple different tasks in parallel, usually 4 to 16, or subdivide a task and execute its parts concurrently. In contrast, a GPU may achieve a high level of parallelism, on the order of thousands of concurrent threads, as long as they all perform the same computation.

Physically based simulation lends itself relatively well to GPU processing when the problem is reduced to solving a sparse set of linear equations at every step. However, when introducing topology changes in the simulated object, the coupling between these linear equations changes and the precomputed data that allows solving them quickly must be updated.

In order to minimize the cost of such updates caused by a cutting operation, we propose a new data structure that avoids any sort of reallocation of GPU memory and that reduces the amount of data that needs to be copied after many small changes are made to it.

6.1.1 Background

Before discussing GPUs specifically, we first summarize the main methods used for performing physical computations for surgery simulation. The simplest method is with a mass-spring system (MSS), where a set of points – the masses, or particles – are connected through springs, which introduce axial forces between the points when they are stretched or compressed [95]. While MSS are easy to implement, it is difficult to choose the right spring stiffness parameters that will accurately simulate the behavior of soft tissue. Finite element methods (FEM) provide a more realistic model of deformable bodies by solving the continuum elasticity equations over a domain divided into elements [14]. They are also referred to as mesh-based methods, since the elements form a mesh. In contrast to FEM, meshless methods solve these equations with a more diffuse discretization of space, where the “elements” are less geometrically defined and may overlap each other [56]. They are also called particle-based

methods. Position-based dynamics (PBD) is another successful approach that may offer the accuracy of FEM or the flexibility of MSS and meshless methods, depending on how the particles are connected together through constraints [82].

These methods ultimately all depend on solving a system of equations. Assembling and solving that system are often the most computationally intensive aspects of the simulation, and therefore the ones that must be targeted for GPU acceleration to achieve the best performance improvement.

Two aspects of GPUs that make them notoriously difficult to use efficiently will be discussed in this paper:

— Operations must be structured in a way that allows hundreds of threads to simultaneously read from memory and perform the same set of computations on the data. This is referred to as the single instruction, multiple threads (SIMT) execution model [148].

— On a GPU, the memory space is different and *not* shared with that of the CPU. Data must be transferred between the two sets of memory whenever the host or GPU make a change that must be read by the other.

Early implementations of solvers were made using a graphics API [96, 117]. It however came with severe restrictions on how the data is structured – using textures rather than arrays – and on the available precision – only 24-bit floating points could be used. The development of general purpose GPU computing (GPGPU) platforms such as CUDA [148] allowed for much more flexibility and complexity in the kinds of solvers that could be implemented.

In the most general sense, a system solver gathers data about a system, such as forces or constraints, and based on this information, determines in what state the system will next be, usually referring to the positions of various nodes forming an object. In our case, the system consists of deformable organs, a surgical tool and any other simulation element that may interact with them.

The simplest solvers are usually explicit ones, which only require to evaluate nodal forces and accelerations at or before the current time, allowing to find the velocities and positions at the end of the time step. They have been used on the GPU in surgery simulation with MSS [97,110], FEM [105,111,112] and meshless methods [113]. Since it requires a low amount of computations, it can relatively easily be combined with a haptic device [108, 112], which requires a high refresh rate, or with expensive computations like cutting, other physical phenomena like melting [113], or direct volume rendering [27].

The main downside to explicit solvers is their stability. Even if they are very fast, they require a certain small time step size that depends on the size of the smallest element in the

simulated object and on its rigidity.

Implicit solvers provide a much higher stability, at the cost of having to solve a non-linear system, which requires more computations. They however allow arbitrarily long time steps. Surgery simulators using that integration scheme usually solve the system using a matrix-free, iterative solver, which requires less memory than a direct one while allowing for topology changes between frames. These solvers have been used to simulate non-linear behavior [119], cutting a tesselated surface embedded in a meshless model [66], in combination with a compliance method for resolving interactions [25] or with constraints to model permanent deformation and cutting [118].

Position-based dynamics offers a way to combine many aspects of a simulation such as deformation, phase changes, liquids et collision detection and response into a single method. It uses a two-phase process – also called prediction-correction scheme – to first move particles freely in time, based on their current speed, then correct their position directly based on a set of constraints. The constraints are solved using a highly parallelizable Gauss-Seidel method. Organ deformation has been simulated using shape-matching constraints [86] or energy constraints [90]. The latter offer a more physically realistic behavior and provide a simple way to cut the object, by removing and adding constraints.

Other methods have been used to simulate deformations on the GPU, like a multigrid iterative solver [41], a direct static solver [102], or an iterative static solver [109]. However, these methods introduce new levels of complexity or rigidity that make them more challenging to use for cutting simulation and, in the case of static solvers, prone to sudden reactions when a user interacts with the simulated objects.

6.1.2 Contributions

In this paper, we propose a simple way to store the graph data describing the relationship between the nodes of an object that is both efficient to access on the GPU and easy to modify from the CPU and to update. We describe it as part of a simulator based on the Element Free Galerkin (EFG) method [56] using an implicit solver, but it is extendable to other methods such as FEM and may be of benefit when using other solvers.

6.2 Method

We first describe the computations needed for determining how a simulated object moves and gets deformed before explaining how we structure the data to efficiently perform these computations on the GPU while allowing for topology changes. While the discussion focuses

on an elasticity problem, our solution can be applied to other simulations that make use of a graph-like structure.

6.2.1 Deformation

We wish to solve the continuum equations of elasticity for a dynamic system:

$$\rho\ddot{\boldsymbol{u}} = \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f}^{ext}, \tag{6.1}$$

where ρ represents mass density, $\boldsymbol{u}$ the displacement field, $\boldsymbol{\sigma}$ the stress and $\boldsymbol{f}^{ext}$ any external forces. Note that since we use a linear elasticity model, $\boldsymbol{\sigma}$ only depends on $\boldsymbol{u}$. After discretization into a set of nodes and linearization, we obtain a system of equations

$$\boldsymbol{M}\ddot{\boldsymbol{u}} = \boldsymbol{K}\boldsymbol{u} + \boldsymbol{f}, \tag{6.2}$$

with $\boldsymbol{M}$ as the mass matrix, $\boldsymbol{K}$ as the stiffness matrix, $\boldsymbol{u}$ the vector of nodal displacements and $\boldsymbol{f}$ the vector of external nodal forces.

For a surgery simulation with haptic interactions, stability is essential. We thus choose to use an implicit dynamic solver, which remains stable for large time steps. Additionally, unlike explicit solvers, the time step is not constrained by the smallest element size, which is difficult to control when arbitrary cutting is allowed. Following the method of [114], we obtain

$$\left(\boldsymbol{M} - \Delta t^2 \boldsymbol{K}\right) \Delta\dot{\boldsymbol{u}} = \Delta t \left(\boldsymbol{f}_0^{elastic} + \boldsymbol{f}_0^{ext}\right), \tag{6.3}$$

where Δt is the length of the time step, $\Delta\dot{\boldsymbol{u}}$ the change in velocity during that time step – the quantity we are trying to determine – and $\boldsymbol{f}_0 = \boldsymbol{f}_0^{elastic} + \boldsymbol{f}_0^{ext}$ the total forces on the object nodes at the beginning of the time step. From the solution to eq. 6.3, we can compute the new nodal displacements as

$$\boldsymbol{u} = \boldsymbol{u}_0 + \Delta t(\dot{\boldsymbol{u}}_0 + \Delta\dot{\boldsymbol{u}}), \tag{6.4}$$

where $\boldsymbol{u}_0$ and $\dot{\boldsymbol{u}}_0$ are respectively the nodal displacement and velocities at the beginning of the time step.

The entries in $\boldsymbol{K}$ are determined by the relationships between nodes, whether they are connected through elements in mesh-based methods or through their influence in meshless methods. Since we are constantly changing these connections by cutting, $\boldsymbol{K}$ also changes constantly, making the use of a precomputed system matrix impossible. Additionally, because

the computation of $\boldsymbol{K}$ is expensive, we prefer to use a matrix-free method, such as conjugate gradient (CG), for solving the linear system. In that case, the main computation becomes the multiplication of $\boldsymbol{K}$ with an arbitrary vector of nodal displacements (or corrections) at each iteration of the solver (see Algorithm 2).

As for the mass matrix term $\boldsymbol{M}$, we lump the object's mass on the nodes, resulting in a diagonal matrix. Its product with a vector can thus be reduced to an element-wise multiplication that can be added to the left-hand side of eq. 6.3.

Algorithm 2 Conjugate gradient algorithm

This algorithm solves the system $\boldsymbol{Ax} = \boldsymbol{b}$ for $\boldsymbol{x}$. In our case, $\boldsymbol{A} = \boldsymbol{K} \cdot k + \boldsymbol{M} \cdot m$ and $\boldsymbol{b}$ is the vector of nodal forces at the beginning of the time step. Two thresholds are also given as criteria for deciding when to stop iterating. Once the while loop terminates, $\boldsymbol{x}$ holds the (approximate) solution.

Initialization: $\boldsymbol{x} \leftarrow \boldsymbol{0}$, $\boldsymbol{p} \leftarrow \boldsymbol{b}$, $\boldsymbol{r} \leftarrow \boldsymbol{b}$, $\rho \leftarrow \boldsymbol{r} \cdot \boldsymbol{r}$, $e \leftarrow \infty$, $d \leftarrow \infty$
while $e > threshold1$ and $d > threshold2$ **do**
  $e \leftarrow \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$
  $\boldsymbol{Ap} \leftarrow \boldsymbol{Kp} \cdot k + \boldsymbol{Mp} \cdot m$
  $d \leftarrow \boldsymbol{p} \cdot \boldsymbol{Ap}$
  $\alpha \leftarrow \frac{\rho}{d}$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \boldsymbol{p}$
  $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha \boldsymbol{Ap}$
  $\beta \leftarrow \frac{\boldsymbol{r} \cdot \boldsymbol{r}}{\rho}$
  $\boldsymbol{p} \leftarrow \boldsymbol{r} + \beta \boldsymbol{p}$
  $\rho \leftarrow \boldsymbol{r} \cdot \boldsymbol{r}$
end while

The computation of the matrix-vector product in Algorithm 2, in particular the $\boldsymbol{Kp}$ product, is driven by the structure that connects the nodes together. For example, with finite elements, the domain may be subdivided into tetrahedra, where each tetrahedron connects four nodes. With the EFG method, which we use in the present work, each cubic integration element combines a set of about eight nodes. However, the computation method is the same regardless of how the elements are formed. Algorithm 3 describes how we compute $\boldsymbol{Kp}$ by looping over the elements, compute their force density, and distribute it to their nodes.

The rest of the conjugate gradient consists of relatively simple vector operations, such as additions and scalar products, which can easily be carried out on the GPU. As pointed out and implemented by [25], for each iteration, only two scalars need to be transmitted to the CPU to determine whether to continue iterating.

The operations presented in this section represent the most intensive part of the simulation and are thus the ones that need to be targeted for a GPU implementation. However, they

Algorithm 3 Computing the $\boldsymbol{Ku}$ product

This algorithm computes the $\boldsymbol{Kp}$ product using $\boldsymbol{p} = \boldsymbol{u}$. Other inputs are the set of elements E, each with a volume V_e and a neighborhood N_e, the shape functions ϕ_n^e for each element e and node n, and the Lamé parameters λ and μ of the simulated material. In that specific case, the result of the multiplication is a vector of nodal elastic forces $\boldsymbol{f}$ (with nodal values $\boldsymbol{f}_n$).

for e ∈ E **do**
  $\nabla \boldsymbol{u}_e \leftarrow \sum_{n \in N_e} \boldsymbol{u}_n \cdot \nabla \phi_n^e$
  $\boldsymbol{\varepsilon}_e \leftarrow \frac{\nabla \boldsymbol{u}_e + \nabla \boldsymbol{u}_e^T}{2}$
  $\boldsymbol{\sigma}_e \leftarrow 2\mu\boldsymbol{\varepsilon} + \lambda tr(\boldsymbol{\varepsilon})\boldsymbol{I}$
  for n ∈ N_e **do**
    $\boldsymbol{f}_n^e \leftarrow -V_e \boldsymbol{\sigma}_e \nabla \phi_n^e$
    $\boldsymbol{f}_n \leftarrow \boldsymbol{f}_n + \boldsymbol{f}_n^e$
  end for
end for

depend on data structures that are modified every time a topology change occurs in the simulated model and must be implemented in a way that is not overly penalized by these changes.

6.2.2 Changing connectivity data

We now describe how the element connectivity data is encoded so that it can be efficiently modified on the GPU. As a general rule for GPU computing (also for optimal cache access on the CPU), we want pieces of data that will be accessed together to be located close to each other in memory.

We perform topology changes using the method presented in [146] and map the surface onto the physical model as in [149]. There are four sets of data, illustrated in Figure 6.1, that are very large and need to be updated every frame:

1. the connectivity graph that assigns a set of nodes to each element; for example, element I would be $\{i \mid i \in a, b, d, e\}$
2. the shape function value for each element-node pair; I = $\left\{\phi_i^I \mid i \in a, b, d, e\right\}$
3. the shape function derivative for each element-node pair; I = $\left\{\nabla\phi_i^I \mid i \in a, b, d, e\right\}$
4. the weights of the mapping between surface vertices and nodes; for example, vertex 3 would be $\{w_i^3 \mid i \in d, e\}$.

They are represented in the diagram of Figure 6.1. The connectivity between nodes and

surface vertices actually uses the same graph as the connectivity between nodes and elements, and thus requires little additional data to update.

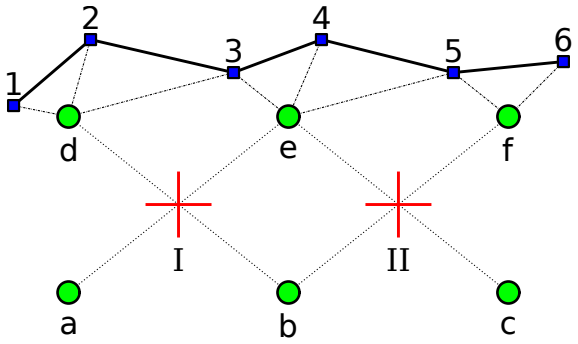


Figure 6.1 Illustration of what the four large 2D data arrays needed for updating an object's state represent. Red crosses are the center of elements, green circles the simulation nodes and blue square are surface vertices. The connectivity links elements to nodes, with shape function values and derivatives associated with each pair. The surface mapping links vertices to nodes, with a weight associated with each pair.

Figure 6.2 illustrates which operations of the simulation are performed on the CPU and which are performed on the GPU. It also shows what data need to be transferred for each of these operations. Names in bold are the 2D arrays discussed in this section.

Two considerations guided our choice of data structure for storing the connectivity and shape functions and mapping: avoid constantly reallocating memory on the GPU and group many small memory transfers into a larger batch – to avoid the relatively large latency associated with each transfer.

These four data sets may be viewed as two-dimensional arrays. For example, the element-node connectivity has a row for each element, containing the identifier of the nodes that are connected to that element. Unlike mesh-based methods, the number of nodes per element may vary, so the rows have uneven lengths.

A memory-efficient way to store such a 2D array, as done for example by [119], would be to have a linear array containing all rows contiguously, with a second array indicating where each row starts in the larger one, and a optionally a third one to store the length of each row. However, with such a structure, if a row increases in length from one frame to the next, all subsequent values in the larger array would need to be shifted. That would be almost equivalent to updating the entire data structure, in addition to having to reallocate memory for it. The cost of a new allocation as well as en entire copy would be noticeable in an application where fast update rates are required.

To avoid having to reallocate memory, even after a change in size for some rows, our solution

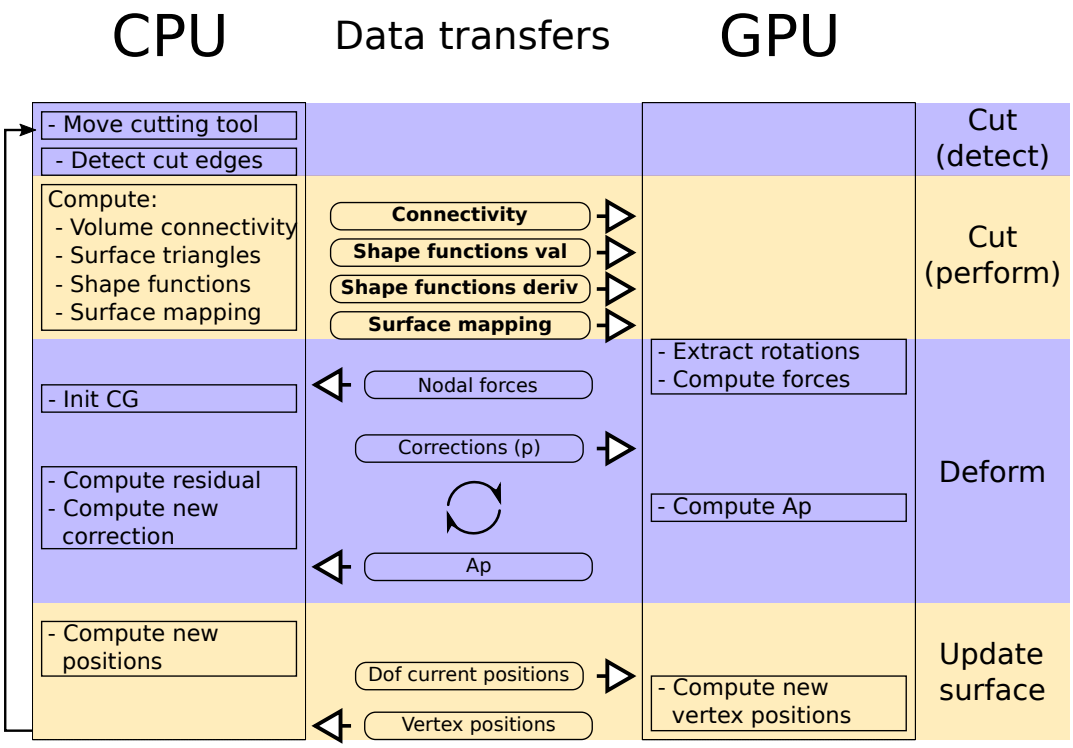


Figure 6.2 Execution steps and data flow of the main simulation loop highlighting which operations are performed on the CPU and on the GPU. Boldface data are the large 2D structures described in this paper.

is to set a maximum row size and allocate a single block that is overall slightly larger than what is strictly needed. To keep track of the actual number of items in each row, we also add a vector that lists the current size of each row. This two-array structure is illustrated in Figure 6.3. Since no elements are added during the simulation, the row size and data block arrays will never need to be resized or reallocated.

This structure is somewhat less flexible than the compact array, because the number of nodes attached to an element can never exceed the allocated maximum row size. However, the criterion that determines whether we need to add nodes in the neighborhood of an element is whether these nodes are coplanar (see [146] for details). The cases where this condition cannot be satisfied when choosing at least 8 neighbors are theoretically rare and will be discussed further in Section 6.3.2.

During a single time step, only a small fraction of all elements will have a new set of neighbor nodes. Similarly for surface changes, only a relatively small number of vertices will need a new mapping onto the volume. To avoid copying the entire data structure and the overhead of many small memory transfers, we divide the data arrays into batches, which can be copied

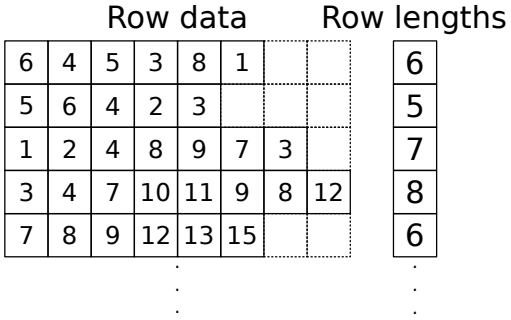


Figure 6.3 Data structure used for major 2D arrays – connectivity data is illustrated here. In the first, large array, rows all occupy the same amount of memory, regardless of the number of elements in them. A secondary array contains the number of elements (length) in every row.

one at a time to GPU memory. The batches are all part of the same memory allocation, only memory transfers are affected by this division. This reduces the total amount of data to be copied every time step while keeping the number of memory transfers low.

6.3 Results

To examine whether the particular data structure described in Section 6.2 allows for an efficient simulation on the GPU and how the performance evolves in different situations, we have implemented the method described in [146] using CUDA and incorporating that structure. All tests whose results are presented in this section were run on a 6-core Intel Core i5-9400F CPU and an Nvidia RTX 2060 GPU.

6.3.1 Performance Comparisons

We first compared the execution time of each simulation update – or time step – of the simulation when run on the CPU only and when using the GPU for computing deformation, in each case using both single and double precision floating-point arithmetic. Figure 6.4 displays the evolution of each update execution time with respect to the number of elements in the simulated model. The scene only contains a torus being partially cut at various volumetric resolutions, with a surface of approximately 50k triangles. The total time displayed includes every aspect of the simulation, which can be categorized into cut detection, cut application, deformation computation and surface position update, as in Figure 6.2.

We can observe in Figure 6.4 that the GPU version is faster at all problem sizes, and that the difference only grows larger as the number of elements increases. The relatively worse

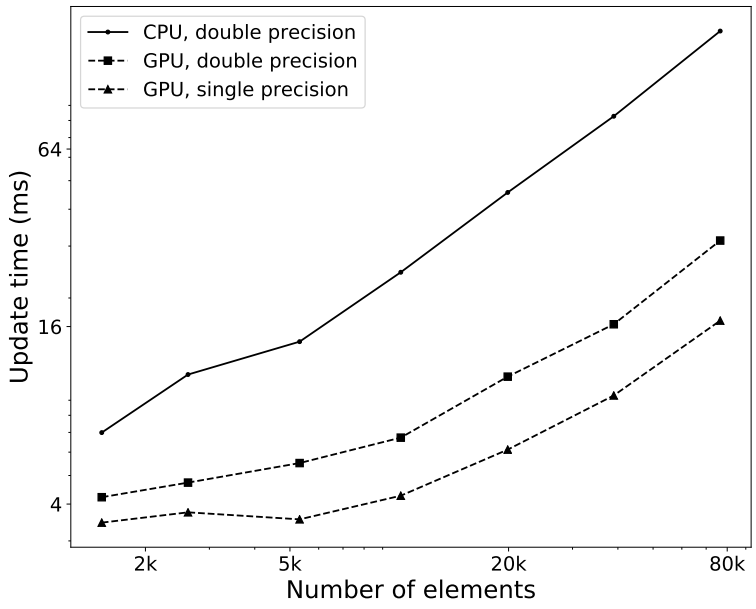


Figure 6.4 Total update time (in ms) for different configurations, with respect to number of elements in physical model.

performance of the GPU version for smaller problem sizes is due to the fact that the 2k-core GPU itself is not used to its full capacity.

Using single precision arithmetic, the GPU version maintains 60 frames per second while cutting a model containing 80k elements, whereas the CPU version reaches that rate at about only 6k elements. CPU performance is only shown for double precision computations, since it was slightly better than single precision performance. At larger sizes, the single-precision GPU version is approximately 45% faster than the double-precision version. This speedup could be improved slightly further by also using the GPU to detect which edges are cut, for both volume and surface. The cutting and surface mapping algorithm however does not lend itself well to a parallel implementation [149].

To determine how the size of the surface mesh affects the performance of the GPU implementation, we also examined how the total computation time evolves with an increasing surface mesh resolution. The results are shown in Figure 6.5, using a physical model of 20k elements that allowed for interactive refresh rates on the CPU. It shows that the surface mesh size barely affects simulation time. This result indicates that the growth of simulation time related to surface operations (cut detection, cut application and position update) is much smaller than that of the time related to volume operations, which is taken mostly by

deformation computation, and to a lesser extent by volume cutting.

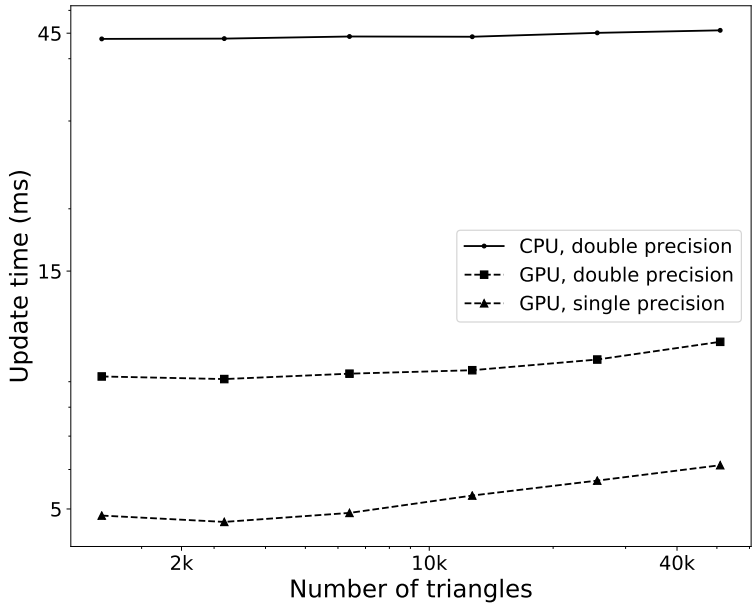


Figure 6.5 Total update time as a function of the number of triangles in the surface mesh. The volumetric model used for these tests has 20k elements, which was the approximate size limit to obtain an interactive update rate on the CPU.

Figure 6.6 displays the proportion of execution time taken by each major step of the main simulation loop. The deformation step, even when running on the GPU, takes the largest proportion with 75% of the update time. Magnoux *et al.* [146] showed that this proportion increases with the number of volume elements, but decreases slightly with a larger number of surface triangles.

Figure 6.7 displays the running times of the same set of simulations as Figure 6.4, for the GPU versions where the splitting of 2D arrays into batches was either activated or not. It shows a regular performance improvement of approximately 7% in single precision and 11% in double precision for all volumetric model sizes. The speed gain is however proportional to the size of the surface mesh (which was of 50k triangles in this test). This indicates that the savings engendered by splitting the 2D arrays are more important for the surface mapping data, which happen to form the largest of all arrays in almost all tested cases – the exception being very refined volumes displayed in very coarse surfaces. In our test cases, splitting the arrays into batches resulted in a reduction of 80 to 90% in the amount of data transferred during each simulation update.

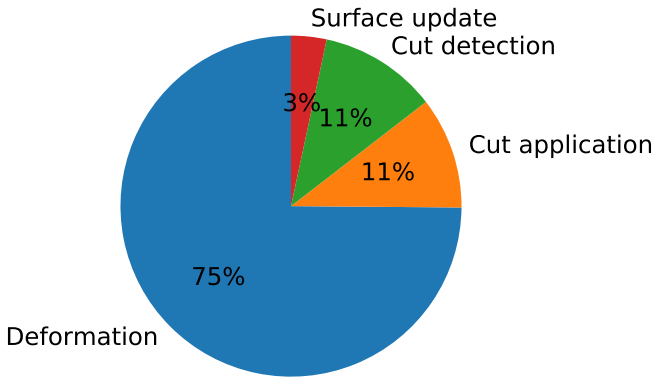


Figure 6.6 Proportion of execution time taken by each major step of the main simulation loop, for the case using single precision floating points on the GPU, with 20k volume elements.

The better gains shown with double precision suggest that a more important proportion of the simulation is spent copying data between RAM and GPU in that case. However, when looking at the distribution of execution time for the deformation step only, shown in Figure 6.8, we can see that making CUDA API calls incurs a relatively large overhead. The overhead remains constant for each kernel launch and memory transfer – regardless of the amount of data – with the memory transfers being responsible for most of that overhead.

Table 6.1 presents an informal comparison of our method with other GPU-based surgery simulators. We achieve a large overall improvement in reported performance among methods that allow cutting operations. However, this does not take into account the fact that the tests in other papers were run on older hardware, or differences in features such as non-linear elastic behavior, other physical phenomena [113], or good contact resolution [25].

6.3.2 Limitations

One obvious limitation of our data structure is that an element may not be connected to more nodes than the maximum allocated. While it has not occurred in any of our test cases, the possibility cannot be ruled out. Allocating more space would mean that a large portion of the structure would simply be empty. A better solution would be to guarantee that a situation requiring more than the maximum number of neighboring nodes can never occur. An approach that could achieve this would be to carefully choose the placement of nodes so that they cannot be coplanar.

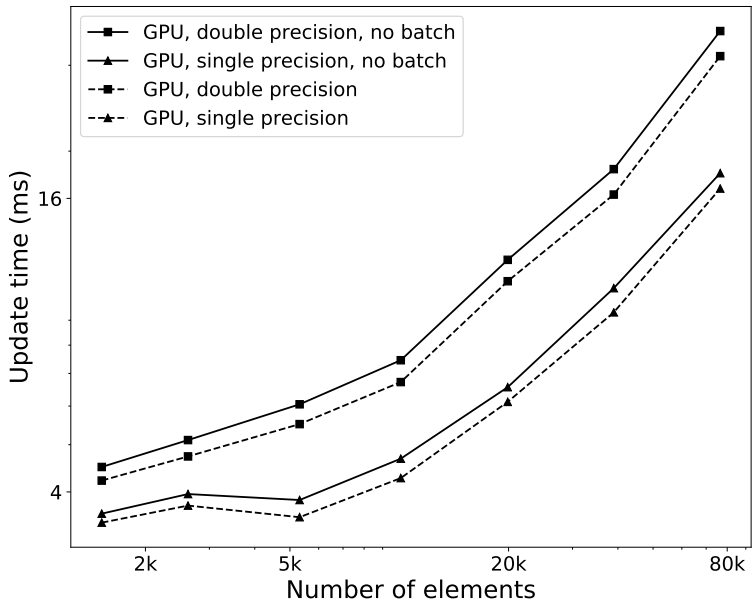


Figure 6.7 Comparison of total update times for the GPU version of the simulation with and without splitting 2D arrays of data into batches, both in single and double precision.

Another potential issue is that the use of single precision arithmetic might result in a difference in the elastic behavior of simulated objects. This would be of particular interest when simulating a large number of elements, which could generate large position values with very small deformations, and very small corrections during individual CG iterations. More specific tests need to be performed to determine whether that is the case.

6.4 Conclusion and Future Work

We have presented a data structure that can be used to describe graph structures in a way that can be read efficiently on graphics hardware, that is modified on the CPU and efficiently updated on the GPU with as little transferred data as possible. Our results show that this structure allows cutting objects simulated with a large number of elements at a sustained high update rate.

Future work will focus on integrating this method into an existing surgery simulator that enables a user to interact with virtual objects through a haptic device and provides highly detailed 3D visual feedback.

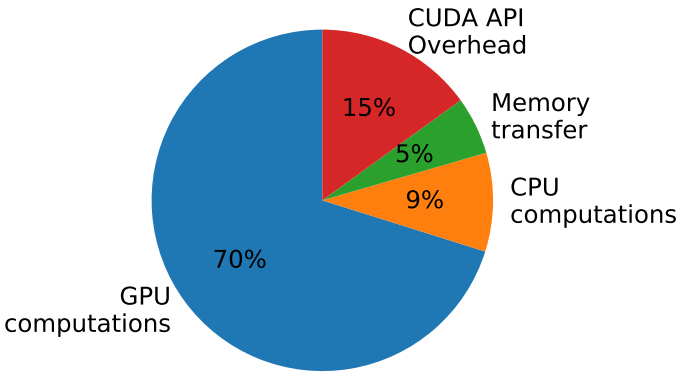


Figure 6.8 Proportion of execution time taken by different aspects of the deformation step, using single precision. Computations take most of the time, mainly on the GPU, while memory transfer and CUDA overhead remain low.

Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) [grant number 501444-16], in collaboration with OSSimTech.

Table 6.1 Summary of method and reported performance for other GPU-based surgery simulators, compared to our method.

Paper	Solver type	Cutting	1000's of nodes	Update rate	Method
Pan [113]	Explicit	Yes	4.9	43	Meshless
Bosman [119]	Implicit	No	12.7	31	Meshless
Pietroni [66]	Implicit	Yes	0.6	15	Meshless
Courtecuisse [25]	Implicit	Yes	0.7	45	FEM
Hou [118]	Implicit	Yes	13	31	FEM
Camara [86]	Gauss-Seidel	No	5.5	79	PBD
Pan [90]	Gauss-Seidel	Yes	1.5	50	PBD
Dick [41]	Multigrid	No	38	62	FEM
Fenz [102]	Static	No	11	20	FEM
Our method	Implicit	Yes	25	60	Meshless

CHAPITRE 7 DISCUSSION GÉNÉRALE

7.1 Travaux complémentaires

Les chapitres 4, 5 et 6 ont présenté le travail fonctionnel accompli au cours de ce projet. Cependant, en nous concentrant uniquement sur ces résultats, nous avons omis ou survolé trop rapidement certains aspects de la méthode qui sont nécessaires pour qu'elle puisse former un simulateur complet. Les sections suivantes discutent un peu plus en détail de ces éléments.

7.1.1 Stabilité

L'utilisation de la corotation semble entraîner des instabilités lorsque des sections trop minces d'un objet sont découpées. Nous avons établi dans la section 4.3.3 qu'il n'est pas possible de couper une tranche d'un objet plus petite que la moitié de la taille d'un élément. Cependant, la méthode de recalage de forme utilisée pour extraire la rotation devient imprécise lorsque l'ensemble de particules dont on cherche la rotation est trop étalé. Cette perte de précision commence à avoir un effet même avec des tranches d'une largeur de deux éléments.

Donc, un ajout important à la méthode de déformation proposée serait un calcul plus précis de la rotation des particules voisine d'un élément, par exemple en l'extrayant directement du champ de déformation. Cette façon de faire aurait probablement également un léger avantage au niveau de la performance étant donné la réduction du nombre d'opérations.

Une autre façon d'améliorer la stabilité serait de compromettre le critère selon lequel l'ajout de coupures n'augmente pas le temps de calcul de la déformation. En permettant une augmentation du nombre de nœuds de calcul, par exemple en adaptant la technique du nœud virtuel à la méthode EFG, il serait possible à la fois d'obtenir une simulation plus stable en garantissant un voisinage régulier autour de chaque élément et de permettre de former des tranches d'objet d'une minceur arbitraire.

Ces améliorations élimineront les restrictions sur l'utilisateur, qui l'empêchent présentement de découper de certaines façons spécifiques, pour lui permettre d'effectuer un découpage complètement libre.

7.1.2 Mappage de surface

Le chapitre 5 décrit un algorithme pour choisir comment chaque sommet de la surface triangulée se déplace avec les particules formant le volume de l'objet. Cet algorithme s'ajoute

à la méthode de découpage décrite dans le chapitre 4. Cependant, il augmente de façon considérable le temps de calcul occupé par l'aspect découpage de la simulation. Par ailleurs, les triangles fournis par la méthode de découpage progressif de la surface sont parfois mal formés, ce qui cause un mauvais calcul de leurs normales, qui ensuite cause un mauvais choix de mappage pour les sommets affectés.

Ces deux problèmes, de performance et d'erreur de mappage, sont mieux abordés de concert puisqu'une solution à l'un pourrait également éliminer l'autre. Pour choisir sur quel élément un sommet s'attache, on considère comme candidates toutes les particules connectées du voisinage et on élimine celles qui se trouvent du côté extérieur du triangle. Si la normale calculée est invalide (dans une direction arbitraire), il est possible d'éliminer toutes les candidates. Il faut ensuite recommencer la recherche en relaxant les critères de sélection.

Une meilleure implantation de l'algorithme pourrait améliorer la performance en évitant des recherches multiples, mais ne règlerait pas le problème d'une mauvaise sélection. Une autre option consisterait à élaborer un algorithme de sélection différent, à la fois plus performant et qui ne serait pas affecté par les entrées données par le générateur de surface. Cependant, étant donné que l'erreur de sélection est causée par la mauvaise qualité des triangles générés, une meilleure piste de solution serait d'effectuer un meilleur contrôle sur la forme des triangles lors de leur génération. Ce travail consisterait donc plutôt à corriger l'implantation que de modifier les méthodes.

7.1.3 Détection de collisions

La détection de collisions est présente implicitement ou explicitement dans tous les articles de cette thèse. Cependant, il s'agit seulement de détecter l'intersection entre le fil de la lame simulée et les arêtes de la connectivité et des triangles des objets déformables. Pour permettre des interactions plus variées, incluant le calcul d'un retour haptique, il est nécessaire de calculer les intersections avec des objets entiers.

Si on désire seulement une interaction avec des outils rigides, la presque totalité des éléments est déjà en place dans l'implantation courante de la méthode. L'intersection entre les triangles de l'objet déformable et ceux de la surface de l'outil peut être calculée directement, de la même façon qu'avec le tranchant de l'outil.

Cependant, pour simuler un plus grand nombre d'objets dans une scène chirurgicale, par exemple plusieurs organes en contact les uns avec les autres, il est nécessaire de détecter les collisions entre les surfaces de chaque paire d'objets. Ce problème est alors beaucoup plus complexe. Pour des objets avec des surfaces détaillées, il devient nécessaire d'utiliser

des structures d'accélération qui subdivisent les objets ou l'espace pour faciliter la détection. Lorsqu'en plus ces surfaces se déforment et sont donc différentes à chaque pas de temps, les structures d'accélération doivent être mises à jour, ce qui peut demander une quantité importante de calculs. La détection de collisions peut alors devenir aussi coûteuse que le calcul de la déformation.

7.1.4 Retour haptique

Nous avons mentionné le retour haptique tout au long de ces travaux, sans toutefois expliquer comment le calculer avec notre méthode. En réalité, cette méthode ne pose pas de difficulté particulière pour calculer un retour d'effort. La seule partie manquante consiste à détecter les bons contacts avec l'outil chirurgical, de la façon mentionnée ci-dessus. Une fois les contacts détectés, il est possible d'appliquer des forces de pénalité pour que les objets se repoussent, ou utiliser la méthode de compliance pour une résolution plus stable des contacts. Dans les deux cas, les forces calculées peuvent être transmises directement à l'utilisateur par l'intermédiaire de l'appareil haptique.

Pour rafraîchir les forces rendues par l'appareil haptique à une fréquence plus élevée (300+ Hz) que le reste de la simulation, il est cependant nécessaire d'extrapoler les variations d'après la force aux pas de temps précédents. Par ailleurs, une attention particulière doit être portée à l'insertion du calcul de force dans la séquence interactive de détection, découpage et déformation.

7.2 Retour sur les objectifs

Nous rappelons ici brièvement les objectifs de recherche avant de détailler comment ils ont été remplis par les travaux des trois chapitres précédents.

7.2.1 Implantation d'une méthode avec particules et découpage

L'objectif le plus essentiel consistait à implanter une méthode d'animation d'un corps déformable qui s'appuie sur la mécanique des milieux continus. À cet objectif s'ajoutait celui de pouvoir modifier la topologie d'un objet simulé par cette méthode d'animation.

Le choix de la méthode EFG nous a donné une plateforme de base à partir de laquelle les différents autres aspects ont pu être développés. Le chapitre 4 présente une implantation réussie de la méthode EFG et détaille l'algorithme permettant de découper un objet. Le chapitre 5 explique ensuite comment cet algorithme s'insère dans un système dynamique.

L'utilisateur contrôle un outil qui se déplace en même temps que les objets déformables et qui modifie ces objets d'après les mouvements, tout en gardant la cohérence des déplacements, déformations et modifications.

7.2.2 Gestion de la surface

Pour avoir un rendu visuel et haptique avec une précision du même ordre que celle de la déformation, nous avons choisi comme approche de développer une représentation explicite de surface qui se combine avec la méthode de déformation. Les objectifs correspondants consistaient à rattacher une représentation séparée de la surface à la représentation volumique de l'objet, ainsi qu'à modifier cette surface tout en maintenant sa conformité avec le volume déformé et découpé.

Le chapitre 4 démontre comment une surface triangulée se conforme aux déformations de l'objet qu'elle représente, et ce avant et après que des coupures aient été introduites dans l'objet. Le chapitre 5 explique ensuite comment préserver cette conformité entre la surface et le volume au fur et à mesure que les deux se font découper par un outil tranchant, tout en se déformant.

7.2.3 Utilisation du GPU

Le dernier objectif prescrit voulait produire une adaptation des méthodes élaborées pour les autres objectifs qui permettrait de les exécuter aussi efficacement que possible sur une carte graphique. La structure présentée au chapitre 6 a permis une implantation efficace des méthodes des chapitres 4 et 5 sur GPU.

7.3 Résultats des travaux de recherche

Les travaux de recherche effectués pour produire cette thèse ont d'abord donné lieu à un publication scientifique conjointe avec J.-N. Brunet [72], suivi des trois publications présentées dans cette thèse, chacune mettant en place une pièce essentielle de la méthode développée. Les travaux initiaux ont mené au choix de la méthode EFG comme base pour le calcul de déformation. Cette méthode produit en effet une animation stable comparée aux autres méthodes particulaires, tout en maintenant une topologie flexible qui se modifie facilement. Elle est également très similaire aux méthodes par éléments finis et permet donc de facilement intégrer les techniques adaptées à ces méthodes, comme la corotation et les différents solveurs.

La première étape à la suite de ce choix consistait à concevoir une méthode pour représenter

la topologie d'un objet explicitement, de façon à pouvoir la modifier tout en conservant la capacité de calculer les déformations (chapitre 4) [146]. Nous avons ensuite conçu un algorithme permettant d'utiliser cette méthode de découpage au sein d'une animation, tout en préservant l'intégrité des différentes étapes de l'animation : déplacement de l'outil, détection des contacts, modification de la topologie, déformation de l'objet (chapitre 5) [149]. La dernière étape du projet visait à démontrer, à partie de la méthode EFG étendue au cours des étapes précédentes, son applicabilité à une exécution sur GPU.

7.4 Contributions

En résumé, cette thèse a présenté les contributions suivantes :

- Une description de la topologie d'un objet selon la méthode EFG. Cette description comprend deux graphes de connectivité entre les particules et entre les particules et les éléments volumiques. (Chapitre 4)
- Un algorithme pour modifier cette topologie et choisir les bons nœuds voisins d'un élément à la suite d'une coupure. (Chapitre 4)
- Une méthode progressive rapide pour produire un maillage sur la surface de l'objet dévoilée par une coupure. Cette méthode réutilise les triangles existants dans la mesure du possible pour en générer seulement un petit nombre, même lorsque l'outil tranchant fait plusieurs petits mouvements. (Chapitre 4)
- Une interaction complètement continue entre un outil tranchant et une surface triangulée, qui inclut la détection des primitives à découper et les modifications qui en suivent. L'outil et la surface peuvent tous deux se déformer durant l'interaction. (Chapitre 5)
- Un ensemble de critères qui permettent de choisir sur quel élément du volume un sommet de la surface doit se calquer pour obtenir la cohérence visuelle du volume déformé et découpé. (Chapitre 5)
- Une structure de donnée qui décrit un graphe et qui permet une implantation GPU efficace de la méthode élaborée durant le projet. Cette structure est applicable à toute méthode de simulation basée sur de tels graphes qui nécessitent des modifications fréquentes. (Chapitre 6)

En plus de cette production scientifique, nos travaux ont donné lieu à une librairie logicielle permettant de décrire un objet déformable pouvant être découpé. Elle fournit également des solveurs et d'autres éléments nécessaires à une simulation. Cette librairie peut être utilisée avec différentes plateformes de simulation, en particulier SOFA et Unreal Engine, ou par elle-même avec un petit exécutable.

CHAPITRE 8 CONCLUSION

La simulation de chirurgie dans un environnement de réalité virtuelle demeure un problème très complexe. Atteindre un réalisme convaincant demande une grande précision dans la description du comportement des organes ainsi que des interactions lors d'une chirurgie simulée.

8.1 Synthèse des travaux

Nous avons abordé le problème particulier du découpage à l'aide d'un outil tranchant, avec pour objectif particulier la mise en place d'une méthode permettant de simuler efficacement et de façon réaliste le comportement d'un objet déformable pendant qu'un scalpel le découpe. Pour y parvenir, nous avons pris comme point de départ une méthode EFG, développé une description simple de la topologie d'un objet et conçu un algorithme pour modifier cette topologie à l'aide d'un plan tranchant. Nous avons ensuite développé une description dynamique du fil de la lame d'un outil et la méthode par laquelle un tel outil coupe un objet en plein mouvement.

Pour accompagner la description physique volumique de l'objet déformable, nous avons inclus une représentation surfacique composée de triangles. Nous avons conçu une méthode complètement progressive pour incorporer dans cette surface les coupures par l'outil au moment où elles se produisent. Afin que le mouvement et la déformation de la surface suivent fidèlement le volume, y compris durant le découpage, nous avons développé un algorithme qui permet de choisir comment chaque sommet se positionne par rapport aux nœuds qui composent l'objet.

Finalement, pour augmenter la résolution des objets déformables simulés et pour permettre une plus grande complexité dans la simulation, nous avons développé une structure particulière pour décrire la topologie qui est bien adaptée aux calculs sur GPU. Cette structure facilite une implantation efficace sur GPU des méthodes développées durant nos travaux.

8.2 Limitations

Les principales limitations de la méthode décrite dans ce document concernent le sentiment de présence d'un utilisateur de cette méthode. Un élément essentiel pour maintenir l'immersion est la stabilité de la simulation. Ainsi, la coupure d'une tranche d'objet trop mince peut donner lieu à des erreurs dans la résolution de la déformation et causer un mouvement spontané de cette tranche. Par ailleurs, des aberrations visuelles peuvent se produire lorsque les sommets de la surface ne sont pas correctement calqués sur le volume. Ce problème

provient plus de la complexité d'implémenter la méthode de génération de surface que d'un défaut dans la méthode même.

8.3 Améliorations futures

Nous avons proposé des pistes de solution pour pallier les limitations de la méthode décrite. En plus de ces améliorations, le prochain objectif pour rendre cette méthode plus complète sera d'intégrer le calcul d'un retour haptique au sein de la méthode dynamique de calcul de découpage, où les différentes étapes – mouvement de l'outil, détection d'intersection, changements à la topologie, calcul des déformations – doivent être soigneusement agencées pour assurer une interaction fluide et réaliste.

RÉFÉRENCES

[1] D. Terzopoulos, J. Platt, A. Barr et K. Fleischer, "Elastically deformable models," dans *ACM Siggraph Computer Graphics*, vol. 21. ACM, 1987, p. 205–214.

[2] K. Miller, G. Joldes, D. Lance et A. Wittek, "Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation," *Communications in Numerical Methods in Engineering*, vol. 23, n°. 2, p. 121–134, févr. 2007.

[3] E. Sifakis et J. Barbič, "FEM Simulation of 3D Deformable Solids : A Practitioner's Guide to Theory, Discretization and Model Reduction," dans *ACM SIGGRAPH 2012 Courses*, ser. SIGGRAPH '12. New York, NY, USA : ACM, 2012, p. 20 :1–20 :50.

[4] C. A. Felippa et B. Haugen, "A unified formulation of small-strain corotational finite elements : I. Theory," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, n°. 21–24, p. 2285–2335, juin 2005.

[5] M. Desbrun, P. Schröder et A. Barr, "Interactive animation of structured deformable objects," dans *Graphics Interface*, vol. 99, 1999, p. 10.

[6] N. Molino, Z. Bao et R. Fedkiw, "A Virtual Node Algorithm for Changing Mesh Topology During Simulation," dans *ACM SIGGRAPH 2005 Courses*, ser. SIGGRAPH '05. New York, NY, USA : ACM, 2005, p. 4.

[7] J. R. Shewchuk, "What is a Good Linear Element ? Interpolation, Conditioning, and Quality Measures," dans *Proceedings of the 11th International Meshing Roundtable*. Ithaca, NY : Sandia National Laboratories, sept. 2002, p. 115–126.

[8] M. Müller et M. Gross, "Interactive virtual materials," dans *Proceedings of Graphics Interface 2004*. Canadian Human-Computer Communications Society, 2004, p. 239–246.

[9] Y.-H. Yeung, J. Crouch et A. Pothen, "Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices," *ACM Transactions on Graphics*, vol. 35, n°. 2, p. 1–17, févr. 2016.

[10] D. Steinemann, M. A. Otaduy et M. Gross, "Fast Arbitrary Splitting of Deforming Objects," dans *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '06. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2006, p. 63–72.

[11] S. Lin, R. Narayan et Y.-S. Lee, "Heterogeneous Deformable Modeling and Topology Modification for Surgical Cutting Simulation with Haptic Interfaces," *Computer-Aided Design and Applications*, vol. 5, n°. 6, p. 877–888, janv. 2008.

[12] B. Ghali et S. Sirouspour, "Nonlinear finite element-based modeling of soft-tissue cutting," dans *2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)*, sept. 2009, p. 141–146.

[13] Y.-J. Lim, J. Hu, C.-Y. Chang et N. Tardella, "Soft Tissue Deformation and Cutting Simulation for the Multimodal Surgery Training," dans *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*, 2006, p. 635–640.

[14] S. Cotin, H. Delingette et N. Ayache, "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," *The Visual Computer*, vol. 16, n°. 8, p. 437–452, déc. 2000.

[15] G. Picinbono, H. Delingette et N. Ayache, "Non-linear anisotropic elasticity for real-time surgery simulation," *Graphical models*, vol. 65, n°. 5, p. 305–321, 2003.

[16] G. Yin, Y. Li, J. Zhang et J. Ni, "Soft Tissue Modeling Using Tetrahedron Finite Element Method in Surgery Simulation," dans *2009 First International Conference on Information Science and Engineering*, déc. 2009, p. 3705–3708.

[17] X. Chen, M. Nakayama, A. Konno, X. Jiang, S. Abiko et M. Uchiyama, "Simulation of surgical dissection using a dynamic deformation model," dans *2010 IEEE/SICE International Symposium on System Integration*, déc. 2010, p. 90–95.

[18] B. Lee, D. C. Popescu et S. Ourselin, "Topology modification for surgical simulation using precomputed finite element models based on linear elasticity," *Progress in Biophysics and Molecular Biology*, vol. 103, n°. 2–3, p. 236–251, déc. 2010.

[19] M. Nakao, K. Minato, N. Kume, S. I. Mori et S. Tomita, "Vertex-preserving Cutting of Elastic Objects," dans *2008 IEEE Virtual Reality Conference*, mars 2008, p. 277–278.

[20] M. Seiler, D. Steinemann, J. Spillmann et M. Harders, "Robust interactive cutting based on an adaptive octree simulation mesh," *The Visual Computer*, vol. 27, n°. 6-8, p. 519–529, 2011.

[21] L. Jeřábková, G. Bousquet, S. Barbier, F. Faure et J. Allard, "Volumetric modeling and interactive cutting of deformable bodies," *Progress in Biophysics and Molecular Biology*, vol. 103, n°. 2–3, p. 217–224, déc. 2010.

[22] S. Delorme, D. Laroche, R. DiRaddo et R. F. Del Maestro, "NeuroTouch : a physics-based virtual simulator for cranial microneurosurgery training," *Neurosurgery*, vol. 71, n°. 1 Suppl Operative, p. 32–42, sept. 2012.

[23] D. Bielser, V. A. Maiwald et M. H. Gross, "Interactive Cuts through 3-Dimensional Soft Tissue," *Computer Graphics Forum*, vol. 18, n°. 3, p. 31–38, sept. 1999.

[24] A. B. Mor et T. Kanade, “Modifying Soft Tissue Models : Progressive Cutting with Minimal New Element Creation,” dans *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2000*, ser. Lecture Notes in Computer Science, S. L. Delp, A. M. DiGoia et B. Jaramaz, édit. Springer Berlin Heidelberg, oct. 2000, p. 598–607.

[25] H. Courtecuisse, H. Jung, J. Allard, C. Duriez, D. Y. Lee et S. Cotin, “GPU-based Real-Time Soft Tissue Deformation with Cutting and Haptic Feedback,” *Progress in Biophysics and Molecular Biology*, vol. 103, n°. 2-3, p. 159–168, déc. 2010.

[26] H. Courtecuisse, J. Allard, P. Kerfriden, S. P. Bordas, S. Cotin et C. Duriez, “Real-time simulation of contact and cutting of heterogeneous soft-tissues,” *Medical image analysis*, vol. 18, n°. 2, p. 394–410, 2014.

[27] S. Li, Q. Zhao, S. Wang, A. Hao et H. Qin, “Interactive deformation and cutting simulation directly using patient-specific volumetric images,” *Computer Animation and Virtual Worlds*, vol. 25, n°. 2, p. 155–169, 2014.

[28] H.-W. Nienhuys, “Cutting in deformable objects,” Thèse de doctorat, Utrecht University, Utrecht, 2003.

[29] M. Wicke, M. Botsch et M. Gross, “A Finite Element Method on Convex Polyhedra,” *Computer Graphics Forum*, vol. 26, n°. 3, p. 355–364, sept. 2007.

[30] S. Martin, P. Kaufmann, M. Botsch, M. Wicke et M. Gross, “Polyhedral finite elements using harmonic basis functions,” dans *Computer Graphics Forum*, vol. 27. Wiley Online Library, 2008, p. 1521–1529.

[31] C. J. Paulus, L. Untereiner, H. Courtecuisse, S. Cotin et D. Cazier, “Virtual cutting of deformable objects based on efficient topological operations,” *The Visual Computer*, vol. 31, n°. 6-8, p. 831–841, 2015.

[32] O. Cakir, R. Yazici et O. Cakir, “Real-time cutting simulation based on stiffness-warped FEM,” dans *2009 24th International Symposium on Computer and Information Sciences*, sept. 2009, p. 721–724.

[33] J. Zhang, L. Gu, X. Li et M. Fang, “An advanced hybrid cutting method with an improved state machine for surgical simulation,” *Computerized Medical Imaging and Graphics*, vol. 33, n°. 1, p. 63–71, janv. 2009.

[34] E. Sifakis, K. G. Der et R. Fedkiw, “Arbitrary Cutting of Deformable Tetrahedralized Objects,” dans *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’07. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2007, p. 73–80.

[35] W. Wu et P. A. Heng, “An improved scheme of an interactive finite element model for 3D soft-tissue cutting and deformation,” *The Visual Computer*, vol. 21, n°. 8-10, p. 707–716, sept. 2005.

[36] G. Debunne, M. Desbrun, A. Barr et M.-P. Cani, “Interactive multiresolution animation of deformable models,” dans *Computer Animation and Simulation’99*. Springer, 1999, p. 133–144.

[37] S. Miyazaki, M. Endo, M. Yamada, J. Hasegawa, T. Yasuda et S. Yokoi, “A deformable fast computation elastic model based on element reduction and reconstruction,” dans *2004 International Conference on Cyberworlds*, nov. 2004, p. 94–99.

[38] W. Hackbusch et S. A. Sauter, “Composite finite elements for the approximation of PDEs on domains with complicated micro-structures,” *Numerische Mathematik*, vol. 75, n°. 4, p. 447–472, févr. 1997.

[39] F. Liehr, T. Preusser, M. Rumpf, S. Sauter et L. O. Schwen, “Composite finite elements for 3D image based computing,” *Computing and Visualization in Science*, vol. 12, n°. 4, p. 171–188, avr. 2009.

[40] M. Nesme, P. G. Kry, L. Jeřábková et F. Faure, “Preserving Topology and Elasticity for Embedded Deformable Models,” dans *ACM SIGGRAPH 2009 Papers*, ser. SIGGRAPH ’09. New York, NY, USA : ACM, 2009, p. 52 :1–52 :9.

[41] C. Dick, J. Georgii et R. Westermann, “A real-time multigrid finite hexahedra method for elasticity simulation using CUDA,” *Simulation Modelling Practice and Theory*, vol. 19, n°. 2, p. 801–816, févr. 2011.

[42] J. Wu, R. Westermann et C. Dick, “Real-Time Haptic Cutting of High-Resolution Soft Tissues,” *Studies in Health Technology and Informatics (Proc. Medicine Meets Virtual Reality 2014)*, vol. 196, p. 469–475, 2014.

[43] S.-Y. Jia, Z.-K. Pan, G.-D. Wang, W.-Z. Zhang et X.-K. Yu, “Stable Real-Time Surgical Cutting Simulation of Deformable Objects Embedded with Arbitrary Triangular Meshes,” *Journal of Computer Science and Technology*, vol. 32, n°. 6, p. 1198–1213, nov. 2017.

[44] J. Wu, C. Dick et R. Westermann, “Efficient collision detection for composite finite element simulation of cuts in deformable bodies,” *The Visual Computer*, vol. 29, n°. 6-8, p. 739–749, juin 2013.

[45] G. M. Turkiyyah, W. B. Karam, Z. Ajami et A. Nasri, “Mesh cutting during real-time physical simulation,” *Computer-Aided Design*, vol. 43, n°. 7, p. 809–819, juill. 2011.

[46] L. M. Vigneron, J. G. Verly et S. K. Warfield, “On Extended Finite Element Method (XFEM) for Modelling of Organ Deformations Associated with Surgical Cuts,” dans

Medical Simulation, ser. Lecture Notes in Computer Science, S. Cotin et D. Metaxas, édit. Springer Berlin Heidelberg, 2004, nº. 3078, p. 134–143.

[47] L. F. Gutiérrez et F. Ramos, "XFEM Framework for Cutting Soft Tissue Including Topological Changes in a Surgery Simulation," dans *Proceedings of the International Conference on Computer Graphics Theory and Applications*, Angers, France, janv. 2010, p. 275–283.

[48] L. Chen, T. Rabczuk, S. P. A. Bordas, G. R. Liu, K. Y. Zeng et P. Kerfriden, "Extended finite element method with edge-based strain smoothing (ESm-XFEM) for linear elastic crack growth," *Computer Methods in Applied Mechanics and Engineering*, vol. 209, p. 250–265, 2012.

[49] L. Jeřábková et T. Kuhlen, "Stable Cutting of Deformable Objects in Virtual Environments Using XFEM," *IEEE Computer Graphics and Applications*, vol. 29, nº. 2, p. 61–71, mars 2009.

[50] C. Quesada, D. González, I. Alfaro, E. Cueto et F. Chinesta, "Computational vademecums for real-time simulation of surgical cutting in haptic environments," *International Journal for Numerical Methods in Engineering*, vol. 108, nº. 10, p. 1230–1247, déc. 2016.

[51] P. Wang, A. A. Becker, I. A. Jones, A. T. Glover, S. D. Benford, C. M. Greenhalgh et M. Vloeberghs, "A virtual reality surgery simulation of cutting and retraction in neurosurgery with force-feedback," *Computer Methods and Programs in Biomedicine*, vol. 84, nº. 1, p. 11–18, oct. 2006.

[52] S. Wang, J. C. Gee et J. Yang, "A framework for craniofacial surgery simulation based on pre-specified target face configurations," dans *2009 IEEE International Symposium on Biomedical Imaging : From Nano to Macro*, juin 2009, p. 1055–1058.

[53] K.-S. Choi, S. Soo et F.-L. Chung, "A virtual training simulator for learning cataract surgery with phacoemulsification," *Computers in Biology and Medicine*, vol. 39, nº. 11, p. 1020–1031, nov. 2009.

[54] M. Desbrun et M.-P. Gascuel, "Smoothed particles : A new paradigm for animating highly deformable bodies," dans *Computer Animation and Simulation'96*. Springer, 1996, p. 61–76.

[55] A. Nealen, M. Müller, R. Keiser, E. Boxerman et M. Carlson, "Physically based deformable models in computer graphics," dans *Computer graphics forum*, vol. 25. Wiley Online Library, 2006, p. 809–836.

[56] T. Belytschko, Y. Y. Lu et L. Gu, "Element-free Galerkin methods," *International journal for numerical methods in engineering*, vol. 37, nº. 2, p. 229–256, 1994.

[57] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross et M. Alexa, “Point Based Animation of Elastic, Plastic and Melting Objects,” dans *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’04. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2004, p. 141–151.

[58] G. R. Liu, G. Y. Zhang, Y. T. Gu et Y. Y. Wang, “A meshfree radial point interpolation method (RPIM) for three-dimensional solids,” *Computational Mechanics*, vol. 36, n°. 6, p. 421–430, nov. 2005.

[59] S. Martin, P. Kaufmann, M. Botsch, E. Grinspun et M. Gross, “Unified simulation of elastic rods, shells, and solids,” *ACM Transactions on Graphics (TOG)*, vol. 29, n°. 4, p. 39, 2010.

[60] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutré et M. Gross, “A unified lagrangian approach to solid-fluid animation,” dans *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* IEEE, 2005, p. 125–148.

[61] B. Solenthaler, J. Schläfli et R. Pajarola, “A unified particle model for fluid–solid interactions,” *Computer Animation and Virtual Worlds*, vol. 18, n°. 1, p. 69–82, févr. 2007.

[62] J. Schläfli, “Simulation of fluid-solid interaction,” Thèse de doctorat, University of Zürich, déc. 2005.

[63] D. Gerszewski, H. Bhattacharya et A. W. Bargteil, “A point-based method for animating elastoplastic solids,” dans *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* ACM, 2009, p. 133–138.

[64] M. Becker, M. Ihmsen et M. Teschner, “Corotated SPH for Deformable Solids.” dans *NPH.* The Eurographics Association, 2009, p. 27–34.

[65] N. Holgate, G. R. Joldes et K. Miller, “Efficient visibility criterion for discontinuities discretised by triangular surface meshes,” *Engineering Analysis with Boundary Elements*, vol. 58, p. 1–6, sept. 2015.

[66] N. Pietroni, F. Ganovelli, P. Cignoni et R. Scopigno, “Splitting cubes : a fast and robust technique for virtual cutting,” *The Visual Computer*, vol. 25, n°. 3, p. 227–239, mars 2009.

[67] D. Steinemann, M. A. Otaduy et M. Gross, “Splitting meshless deforming objects with explicit surface tracking,” *Graphical Models*, vol. 71, n°. 6, p. 209–220, nov. 2009.

[68] Y. Peng, Q. Li, Y. Yan et Q. Wang, “Real-time deformation and cutting simulation of cornea using point based method,” *Multimedia Tools and Applications*, vol. 78, n°. 2, p. 2251–2268, janv. 2019.

[69] W. Si, J. Lu, X. Liao et Q. Wang, "Towards Interactive Progressive Cutting of Deformable Bodies via Phyxel-Associated Surface Mesh Approach for Virtual Surgery," *IEEE Access*, vol. 6, p. 32 286–32 299, 2018.

[70] R. Aras, Y. Shen et M. Audette, "An analytic meshless enrichment function for handling discontinuities in interactive surgical simulation," *Advances in Engineering Software*, vol. 102, p. 40–48, déc. 2016.

[71] A. Horton, A. Wittek, G. R. Joldes et K. Miller, "A meshless Total Lagrangian explicit dynamics algorithm for surgical simulation," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 26, nº. 8, p. 977–998, 2010.

[72] J.-N. Brunet, V. Magnoux, B. Ozell et S. Cotin, "Corotated meshless implicit dynamics for deformable bodies," dans *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, Pilsen, Czech Republic, mai 2019, p. 91–100.

[73] H. Jung et D. Y. Lee, "Real-time cutting simulation of meshless deformable object using dynamic bounding volume hierarchy," *Computer Animation and Virtual Worlds*, vol. 23, nº. 5, p. 489–501, sept. 2012.

[74] X. Jin, G. R. Joldes, K. Miller, K. H. Yang et A. Wittek, "Meshless algorithm for soft tissue cutting in surgical simulation," *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 17, nº. 7, p. 800–811, mai 2014.

[75] Q. Cheng, P. X. Liu, P. Lai, S. Xu et Y. Zou, "A Novel Haptic Interactive Approach to Simulation of Surgery Cutting Based on Mesh and Meshless Models," *Journal of Healthcare Engineering*, vol. 2018, p. 1–16, 2018.

[76] M. R. Dehghan, A. Rahimi, H. A. Talebi et M. Zareinejad, "A three-dimensional large deformation model for soft tissue using meshless method," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 12, nº. 2, p. 241–253, juin 2016.

[77] Y. Zou, P. X. Liu, Q. Cheng, P. Lai et C. Li, "A New Deformation Model of Biological Tissue for Surgery Simulation," *IEEE Transactions on Cybernetics*, vol. 47, nº. 11, p. 3494–3503, nov. 2017.

[78] A.-D. Cheng, M. A. Golberg, E. J. Kansa et G. Zammito, "Exponential convergence and H-c multiquadric collocation method for partial differential equations," *Numerical Methods for Partial Differential Equations*, vol. 19, nº. 5, p. 571–594, 2003.

[79] S. De, M. Manivannan, J. Kim, M. A. Srinivasan et D. Rattner, "Multimodal simulation of laparoscopic Heller myotomy using a meshless technique," *Studies in health technology and informatics*, p. 127–132, 2002.

[80] S. De et Y.-J. Lim, “Interactive Surgical Simulation Using a Meshfree Computational Method,” dans *Computational Modeling in Biomechanics*, S. De, F. Guilak et M. M. R. K, édit. Springer Netherlands, 2010, p. 409–433.

[81] S. Banihani, T. Rabczuk et T. Almomani, “POD for Real-Time Simulation of Hyperelastic Soft Biological Tissue Using the Point Collocation Method of Finite Spheres,” *Mathematical Problems in Engineering*, vol. 2013, déc. 2013.

[82] M. Müller, B. Heidelberger, M. Hennix et J. Ratcliff, “Position based dynamics,” *Journal of Visual Communication and Image Representation*, vol. 18, n°. 2, p. 109–118, avr. 2007.

[83] M. Macklin, M. Müller, N. Chentanez et T.-Y. Kim, “Unified Particle Physics for Real-Time Applications,” *ACM Transactions on Graphics*, vol. 33, n°. 4, p. 104, 2014.

[84] J. Bender, M. Müller, M. A. Otaduy, M. Teschner et M. Macklin, “A Survey on Position-Based Simulation Methods in Computer Graphics,” dans *Computer graphics forum*, vol. 33. Wiley Online Library, 2014, p. 228–251.

[85] M. Müller, B. Heidelberger, M. Teschner et M. Gross, “Meshless deformations based on shape matching,” *ACM Transactions on Graphics (TOG)*, vol. 24, n°. 3, p. 471–478, 2005.

[86] M. Camara, E. Mayer, A. Darzi et P. Pratt, “Soft tissue deformation for surgical simulation : a position-based dynamics approach,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 11, n°. 6, p. 919–928, juin 2016.

[87] J. Pan, S. Yan, H. Qin et A. Hao, “Real-time dissection of organs via hybrid coupling of geometric metaballs and physics-centric mesh-free method,” *The Visual Computer*, vol. 34, n°. 1, p. 105–116, janv. 2018.

[88] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl et H.-P. Seidel, “Laplacian surface editing,” dans *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP ’04*. Nice, France : ACM Press, 2004, p. 175.

[89] J. Bender, D. Koschier, P. Charrier et D. Weber, “Position-based simulation of continuous materials,” *Computers & Graphics*, vol. 44, p. 1–10, nov. 2014.

[90] J. Pan, J. Bai, X. Zhao, A. Hao et H. Qin, “Real-time haptic manipulation and cutting of hybrid soft tissue models by extended position-based dynamics,” *Computer Animation and Virtual Worlds*, vol. 26, n°. 3-4, p. 321–335, 2015.

[91] S. Bouaziz, S. Martin, T. Liu, L. Kavan et M. Pauly, “Projective dynamics : fusing constraint projections for fast simulation,” *ACM Transactions on Graphics (TOG)*, vol. 33, n°. 4, p. 154, 2014.

[92] K. Qian, T. Jiang, M. Wang, X. Yang et J. Zhang, "Energized soft tissue dissection in surgery simulation," *Computer Animation and Virtual Worlds*, vol. 27, n°. 3-4, p. 280–289, 2016.

[93] Y. Peng, Y. Ma, Y. Wang et J. Shan, "The application of interactive dynamic virtual surgical simulation visualization method," *Multimedia Tools and Applications*, vol. 76, n°. 23, p. 25 197–25 214, déc. 2017.

[94] E. Sifakis et J. Barbič, "Finite Element Method Simulation of 3D Deformable Solids," *Synthesis Lectures on Visual Computing*, vol. 7, n°. 3, p. 1–69, oct. 2015.

[95] G. Bianchi, M. Harders et G. Székely, "Mesh topology identification for mass-spring models," dans *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2003, p. 50–58.

[96] J. Georgii et R. Westermann, "Mass-spring systems on the GPU," *Simulation Modelling Practice and Theory*, vol. 13, n°. 8, p. 693–702, nov. 2005.

[97] D. Zerbato, D. Baschirotto, D. Baschirotto, D. Botturi et P. Fiorini, "GPU-based physical cut in interactive haptic simulations," *International Journal of Computer Assisted Radiology and Surgery*, vol. 6, n°. 2, p. 265–272, mars 2011.

[98] A. Duysak, J. J. Zhang et V. Ilankovan, "Efficient modelling and simulation of soft tissue deformation using mass-spring systems," dans *International Congress Series*, vol. 1256. Elsevier, 2003, p. 337–342.

[99] G. Bianchi, B. Solenthaler, G. Székely et M. Harders, "Simultaneous topology and stiffness identification for mass-spring models based on fem reference deformations," dans *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2004, p. 293–301.

[100] V. Baudet, M. Beuve, F. Jaillet, B. Shariat et F. Zara, "Integrating tensile parameters in hexahedral mass-spring system for simulation," dans *WSCG proceedings*. Václav Skala - UNION Agency, 2009.

[101] O. Jarrousse, *Modified mass-spring system for physically based deformation modeling*. KIT Scientific Publishing, 2014, vol. 14.

[102] W. Fenz et J. Dirnberger, "Real-time surgery simulation of intracranial aneurysm clipping with patient-specific geometries and haptic feedback," dans *SPIE Medical Imaging*, vol. 9415. International Society for Optics and Photonics, 2015, p. 94 150H–94 150H–10.

[103] G. Debunne, M. Desbrun, M.-P. Cani et A. H. Barr, "Dynamic real-time deformations using space & time adaptive sampling," dans *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, p. 31–36.

[104] H.-x. Wang, A. m. Hao, D. Li et L. Xue-mei, "Real-time cutting method for soft tissue based on TLED algorithm," dans *2010 2nd International Conference on Computer Engineering and Technology*, vol. 3, avr. 2010, p. V3–393–V3–396.

[105] Z. A. Taylor, O. Comas, M. Cheng, J. Passenger, D. J. Hawkes, D. Atkinson et S. Ourselin, "Modelling anisotropic viscoelasticity for real-time soft tissue simulation," dans *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2008, p. 703–710.

[106] G. Y. Zhang, A. Wittek, G. R. Joldes, X. Jin et K. Miller, "A three-dimensional nonlinear meshfree algorithm for simulating mechanical responses of soft tissue," *Engineering Analysis with Boundary Elements*, vol. 42, p. 60–66, mai 2014.

[107] Y. Zhang, Z. Yuan, Y. Ding, J. Zhao, Z. Duan et M. Sun, "Real Time Simulation of Tissue Cutting Based on GPU and CUDA for Surgical Training," dans *2010 International Conference on Biomedical Engineering and Computer Science*, avr. 2010, p. 1–4.

[108] R. J. Lapeer, P. D. Gasson et V. Karri, "A Hyperelastic Finite-Element Model of Human Skin for Interactive Real-Time Surgical Simulation," *IEEE Transactions on Biomedical Engineering*, vol. 58, n°. 4, p. 1013–1022, avr. 2011.

[109] G. R. Joldes, A. Wittek et K. Miller, "An adaptive dynamic relaxation method for solving nonlinear finite element problems. Application to brain shift estimation," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 27, n°. 2, p. 173–185, févr. 2011.

[110] Z. Y. Yuan, Y. H. Ding, Y. Y. Zhang et J. H. Zhao, "Real-Time Simulation of Tissue Cutting with CUDA Based on GPGPU," *Advanced Materials Research*, vol. 121-122, p. 154–161, 2010.

[111] O. Comas, Z. A. Taylor, J. Allard, S. Ourselin, S. Cotin et J. Passenger, "Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA," dans *Biomedical Simulation*. Springer, Berlin, Heidelberg, juill. 2008, p. 28–39.

[112] S. Yibo, X. Hui et Y. Dehai, "Improvements of GPU Implementation of Nonlinear Soft Tissue Deformation with CHAI 3D," dans *3rd International Conference on Multimedia Technology (ICMT-13)*. Atlantis Press, nov. 2013, p. 1196–1203.

[113] J. Pan, Y. Yang, Y. Gao, H. Qin et Y. Si, "Real-time simulation of electrocautery procedure using meshfree methods in laparoscopic cholecystectomy," *The Visual Computer*, vol. 35, n°. 6-8, p. 861–872, juin 2019.

[114] D. Baraff et A. Witkin, “Large steps in cloth simulation,” dans *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, p. 43–54.

[115] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik et others, “Sofa : A multi-model framework for interactive physical simulation,” dans *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*. Springer, 2012, p. 283–321.

[116] H. Courtecuisse, Y. Adagolodjo, H. Delingette et C. Duriez, “Haptic rendering of hyperelastic models with friction,” dans *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, sept. 2015, p. 591–596.

[117] W. Wu et P. A. Heng, “A hybrid condensed finite element model with GPU acceleration for interactive 3D soft tissue cutting,” *Computer Animation and Virtual Worlds*, vol. 15, n°. 3-4, p. 219–227, juill. 2004.

[118] W. Hou, P. X. Liu et M. Zheng, “A new model of soft tissue with constraints for interactive surgical simulation,” *Computer Methods and Programs in Biomedicine*, vol. 175, p. 35–43, juill. 2019.

[119] J. Bosman, C. Duriez et S. Cotin, “Connective tissues simulation on GPU,” dans *VRIPHYS 13 : 10th Workshop on Virtual Reality Interaction and Physical Simulation*. Eurographics Association, 2013, p. 41–50.

[120] G. Saupin, C. Duriez, S. Cotin et L. Grisoni, “Efficient Contact Modeling using Compliance Warping,” dans *Computer Graphics International*, Istanbul, 2008.

[121] J. Wu, C. Dick et R. Westermann, “Interactive High-Resolution Boundary Surfaces for Deformable Bodies with Changing Topology,” dans *VRIPHYS*. The Eurographics Association, 2011, p. 29–38.

[122] D. Steinemann, M. Harders, M. Gross et G. Szekely, “Hybrid Cutting of Deformable Solids,” dans *IEEE Virtual Reality Conference (VR 2006)*, mars 2006, p. 35–42.

[123] W. Shi, P. X. Liu et M. Zheng, “Cutting procedures with improved visual effects and haptic interaction for surgical simulation systems,” *Computer Methods and Programs in Biomedicine*, vol. 184, p. 105270, févr. 2020.

[124] Y. Li, D. Lee, C. Lee, J. Lee, S. Lee, J. Kim, S. Ahn et J. Kim, “Surface embedding narrow volume reconstruction from unorganized points,” *Computer Vision and Image Understanding*, vol. 121, p. 100–107, avr. 2014.

[125] J. Luo, S. Xu et S. Jiang, “A novel hybrid rendering approach to soft tissue cutting in surgical simulation,” dans *2015 8th International Conference on Biomedical Engineering and Informatics (BMEI)*, oct. 2015, p. 270–274.

[126] Y. Zou, P. X. Liu, D. Wu, X. Yang et S. Xu, "Point Primitives Based Virtual Surgery System," *IEEE Access*, vol. 7, p. 46 306–46 316, 2019.

[127] C. Zhu, Y. Xiong, K. Xu et P. Shi, "Fast cutting simulations with underlying lattices," dans *2013 6th International Conference on Biomedical Engineering and Informatics*, déc. 2013, p. 283–289.

[128] M. Kallmann, H. Bieri et D. Thalmann, "Fully dynamic constrained delaunay triangulations," dans *Geometric modeling for scientific visualization*. Springer, 2004, p. 241–257.

[129] H. Jung, S. Cotin, C. Duriez, J. Allard et D. Y. Lee, "High Fidelity Haptic Rendering for Deformable Objects Undergoing Topology Changes," dans *Haptics : Generating and Perceiving Tangible Sensations*. Springer, Berlin, Heidelberg, juill. 2010, p. 262–268.

[130] M. C. Lin et M. Otaduy, *Haptic Rendering : Foundations, Algorithms and Applications*. Natick, MA, USA : A. K. Peters, Ltd., 2008.

[131] J. Barbič et D. L. James, "Six-DoF Haptic Rendering of Contact Between Geometrically Complex Reduced Deformable Models," *IEEE Transactions on Haptics*, vol. 1, nº. 1, p. 39–52, janv. 2008.

[132] J. J. Pan, J. Chang, X. Yang, J. J. Zhang, T. Qureshi, R. Howell et T. Hickish, "Graphic and haptic simulation system for virtual laparoscopic rectum surgery," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 7, nº. 3, p. 304–317, sept. 2011.

[133] G. Cirio, M. Marchal, M. A. Otaduy et A. Lécuyer, "Six-dof haptic interaction with fluids, solids, and their transitions," dans *World Haptics Conference (WHC), 2013*. IEEE, 2013, p. 157–162.

[134] X.-J. He et K.-S. Choi, "Stable Haptic Rendering For Physics Engines Using Inter-Process Communication and Remote Virtual Coupling," *International Journal of Advanced Computer Science and Applications*, vol. 4, nº. 1, 2013.

[135] K. Qian, J. Bai, X. Yang, J. Pan et J. Zhang, "Essential techniques for laparoscopic surgery simulation," *Computer Animation and Virtual Worlds*, 2016.

[136] C. Dick, J. Georgii et R. Westermann, "A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, nº. 11, p. 1663–1675, nov. 2011.

[137] Y. Tai, L. Wei, H. Zhou, J. Peng, J. Shi, Q. Li et S. Nahavandi, "Development of Haptic-Enabled Virtual Reality Simulator for Video-Assisted Thoracoscopic Right Upper Lobectomy," dans *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Miyazaki, Japan : IEEE, oct. 2018, p. 3010–3015.

[138] A. Alaraj, C. J. Luciano, D. P. Bailey, A. Elsenousi, B. Z. Roitberg, A. Bernardo, P. P. Banerjee et F. T. Charbel, "Virtual Reality Cerebral Aneurysm Clipping Simulation With Real-Time Haptic Feedback," *Neurosurgery*, p. 1, janv. 2015.

[139] K. Malukhin et K. Ehmann, "Mathematical Modeling and Virtual Reality Simulation of Surgical Tool Interactions With Soft Tissue : A Review and Prospective," *Journal of Engineering and Science in Medical Diagnostics and Therapy*, vol. 1, nº. 2, p. 020802, mars 2018.

[140] J. Wu, R. Westermann et C. Dick, "A survey of physically based simulation of cuts in deformable bodies," dans *Computer Graphics Forum*, vol. 34. Wiley Online Library, 2015, p. 161–187.

[141] M. Agus, A. Giachetti, E. Gobbetti, G. Zanetti et A. Zorcolo, "Real-time haptic and visual simulation of bone dissection," *Presence : Teleoperators and Virtual Environments*, vol. 12, nº. 1, p. 110–122, 2003.

[142] Y. Kim, L. Kim, D. Lee, S. Shin, H. Cho, F. Roy et S. Park, "Deformable mesh simulation for virtual laparoscopic cholecystectomy training," *The Visual Computer*, vol. 31, nº. 4, p. 485–495, 2015.

[143] D. Zerbato et P. Fiorini, "A unified representation to interact with simulated deformable objects in virtual environments," dans *2016 IEEE International Conference on Robotics and Automation (ICRA)*, mai 2016, p. 2710–2717.

[144] F. Faure, B. Gilles, G. Bousquet et D. K. Pai, "Sparse Meshless Models of Complex Deformable Solids," dans *ACM SIGGRAPH 2011 Papers*, ser. SIGGRAPH '11. New York, NY, USA : ACM, 2011, p. 73 :1–73 :10.

[145] R. Luo, W. Xu, H. Wang, K. Zhou et Y. Yang, "Physics-Based Quadratic Deformation Using Elastic Weighting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, nº. 12, p. 3188–3199, déc. 2018.

[146] V. Magnoux et B. Ozell, "Real-time visual and physical cutting of a meshless model deformed on a background grid," *Computer Animation and Virtual Worlds*, p. e1929, juin 2020.

[147] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garments," dans *Computer Animation and Simulation '97*, ser. Eurographics, D. Thalmann et M. van de Panne, édit. Vienna : Springer, 1997, p. 177–189.

[148] E. Lindholm, J. Nickolls, S. Oberman et J. Montrym, "NVIDIA Tesla : A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, nº. 2, p. 39–55, mars 2008.

[149] V. Magnoux et B. Ozell, “Dynamic Cutting of a Meshless Model for Interactive Surgery Simulation (à paraître),” dans *Salento AVR 2020 7th International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, Lecce, Italy, sept. 2020.

ANNEXE A APPROXIMATION MLS

On cherche à déterminer des fonctions de forme ϕ telles que, à un certain point $\boldsymbol{x}_I$, l'approximation $\boldsymbol{u}^h$ du déplacement soit donnée par

$$\boldsymbol{u}^h(\boldsymbol{x}_I) = \sum_i^{N_I} \phi_i(\boldsymbol{x}_I)\boldsymbol{u}(\boldsymbol{x}_i), \tag{A.1}$$

où $\boldsymbol{x}_i$ est la position courante du nœud i, $\boldsymbol{u}(\boldsymbol{x}_i)$ son déplacement, et $\phi_i(\boldsymbol{x}_I)$ la valeur de la fonction de forme du nœud i à la position $\boldsymbol{x}_I$. On considère que le point $\boldsymbol{x}_I$ est entouré de N_I nœuds $1..N_I$.

On définit les fonctions de formes affectant $\boldsymbol{x}_I$ par

$$\begin{aligned} \boldsymbol{u}^h(\boldsymbol{x}_I) &= \sum_i^{N_I} \phi_i(\boldsymbol{x}_I)\boldsymbol{u}(\boldsymbol{x}_i) \\ &= \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{a}(\boldsymbol{x}_I), \end{aligned} \tag{A.2}$$

où $\boldsymbol{p}(\boldsymbol{x})$ est une base de dimension M et $\boldsymbol{a}(\boldsymbol{x}_I)$ sont les coefficients qui s'appliquent aux composantes de cette base pour obtenir $\boldsymbol{u}^h(\boldsymbol{x}_I)$. Étant donné que la valeur de ces coefficients est calculée uniquement au point $\boldsymbol{x}_I$, on les dénote également $\boldsymbol{a}_I$. De plus, puisque le champ de déplacement $\boldsymbol{u}$ est uniquement connu aux points $\boldsymbol{x}_i$, on dénote les déplacements nodaux $\boldsymbol{u}_i$.

La base peut être

$$\boldsymbol{p}^T(\boldsymbol{x}) = \begin{bmatrix} 1 & x & y & z \end{bmatrix}$$

pour des fonctions de forme linéaires, ou

$$\boldsymbol{p}^T(\boldsymbol{x}) = \begin{bmatrix} 1 & x & y & z & x^2 & y^2 & z^2 & xy & xz & yz \end{bmatrix}$$

pour des fonctions quadratiques.

Pour obtenir les coefficients $\boldsymbol{a}_I$, on minimise la somme pondérée des carrés de la différence entre la valeur cherchée – le déplacement $\boldsymbol{u}_I$ – et les déplacements aux points voisins $\boldsymbol{x}_i$. **Note :** On considère les points $1..N_I$ dans le voisinage de $\boldsymbol{x}_I$ pour déterminer la valeur de $\phi_i(\boldsymbol{x}_I)$.

$$J = \sum_{i=1}^{N_I} w(\boldsymbol{x}_I - \boldsymbol{x}_i) \left[\boldsymbol{p}^T(\boldsymbol{x}_i)\boldsymbol{a}_I - \boldsymbol{u}_i\right], \tag{A.3}$$

où la valeur $w(\boldsymbol{x}_I - \boldsymbol{x}_i)$, souvent représentée par $w_I(\boldsymbol{x}_i)$, est une fonction de pondération (ou *kernel*) dont la valeur est positive à l'intérieur du rayon d'influence de $\boldsymbol{x}_I$ et nulle partout ailleurs.

Le minimum est atteint lorsque J est zéro. On obtient alors un système linéaire

$$\boldsymbol{A}_I \boldsymbol{a}_I = \boldsymbol{B} \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_{N_I} \end{bmatrix}^T, \tag{A.4}$$

où

$$\boldsymbol{A}_{M\times M} = \sum_i^{N_I} w_I(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i)\boldsymbol{p}^T(\boldsymbol{x}_i) \tag{A.5}$$

$$\boldsymbol{B}_{M\times N_I} = \begin{bmatrix} w_I(\boldsymbol{x}_1)\boldsymbol{p}(\boldsymbol{x}_1) & \dots & w_I(\boldsymbol{x}_N)\boldsymbol{p}(\boldsymbol{x}_N) \end{bmatrix}, \tag{A.6}$$

et M est la dimension de la base $\boldsymbol{p}$. $\boldsymbol{A}$ est la matrice de moment du voisinage de $\boldsymbol{x}_I$.

On se retrouve avec

$$\boldsymbol{a}_I = \boldsymbol{A}_I^{-1} \boldsymbol{B} \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_{N_I} \end{bmatrix}^T \tag{A.7}$$

En partant de la définition initiale A.2, on a

$$\sum_i^{N_I} \phi_i(\boldsymbol{x}_I)\boldsymbol{u}_i = \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{a}_I \tag{A.8}$$

$$= \boldsymbol{p}^T(\boldsymbol{x}_I) \left(\boldsymbol{A}_I^{-1} \boldsymbol{B} \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_{N_I} \end{bmatrix}^T \right) \tag{A.9}$$

$$= \begin{bmatrix} \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_{N_I} \end{bmatrix}^T \tag{A.10}$$

$$= \sum_i^{N_I} \begin{bmatrix} \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B} \end{bmatrix}_i \boldsymbol{u}_i. \tag{A.11}$$

La valeur de $\phi_i(\boldsymbol{x}_I)$ est donc la i^{e} composante du vecteur rangée de l'équation A.10 :

$$\phi_i(\boldsymbol{x}_I) = \begin{bmatrix} \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B} \end{bmatrix}_i \tag{A.12}$$

$$= \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1} w_I(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i) \tag{A.13}$$

Pour obtenir le gradient des fonctions de forme, on dérive le long de chaque direction. En x,

on a

$$\phi_{i,x}(\boldsymbol{x}_I) = \left(\boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B}\right)_{,x} \tag{A.14}$$

$$= \boldsymbol{p}_{,x}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B} + \boldsymbol{p}^T(\boldsymbol{x}_I)\left(\boldsymbol{A}_I^{-1}\boldsymbol{B}\right)_{,x} \tag{A.15}$$

$$= \boldsymbol{p}_{,x}^T(\boldsymbol{x}_I)\boldsymbol{A}_I^{-1}\boldsymbol{B} + \boldsymbol{p}^T(\boldsymbol{x}_I)\left(\boldsymbol{A}_{I,x}^{-1}\boldsymbol{B} + \boldsymbol{A}_I^{-1}\boldsymbol{B}_{,x}\right) \tag{A.16}$$

En définissant le gradient d'une valeur estimée par un kernel comme le gradient du kernel multiplié par cette valeur, on obtient

$$\nabla\left(w_I(\boldsymbol{x}_i)\boldsymbol{p}^T(\boldsymbol{x}_i)\right) = \left(\nabla w_I(\boldsymbol{x}_i)\right)\boldsymbol{p}^T(\boldsymbol{x}_i), \tag{A.17}$$

ou, le long de chaque axe :

$$\left(w_I(\boldsymbol{x}_i)\boldsymbol{p}^T(\boldsymbol{x}_i)\right)_{,x} = w_{I,x}(\boldsymbol{x}_i)\boldsymbol{p}^T(\boldsymbol{x}_i). \tag{A.18}$$

Les dérivées de $\boldsymbol{A}^{-1}$ et $\boldsymbol{B}$ sont données par

$$\boldsymbol{A}_{I,x}^{-1} = -\boldsymbol{A}_I^{-1}\boldsymbol{A}_{I,x}\boldsymbol{A}_I^{-1} \tag{A.19}$$

$$\boldsymbol{B}_{i,x} = w_{I,x}(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_i). \tag{A.20}$$

On obtient comme formule finale pour la dérivée d'une fonction de forme

$$\begin{aligned}
\phi_{i,x}(\boldsymbol{x}_I) &= \left[\boldsymbol{p}_{,x}^T(\boldsymbol{x}_I)\boldsymbol{A}^{-1}w_{I,x} + \boldsymbol{p}^T(\boldsymbol{x}_I)\left(\boldsymbol{A}_{,x}^{-1}w_I(\boldsymbol{x}_i) + \boldsymbol{A}^{-1}w_{I,x}(\boldsymbol{x}_i)\right)\right]\boldsymbol{p}(\boldsymbol{x}_i) \\
\phi_{i,y}(\boldsymbol{x}_I) &= \left[\boldsymbol{p}_{,y}^T(\boldsymbol{x}_I)\boldsymbol{A}^{-1}w_{I,y} + \boldsymbol{p}^T(\boldsymbol{x}_I)\left(\boldsymbol{A}_{,y}^{-1}w_I(\boldsymbol{x}_i) + \boldsymbol{A}^{-1}w_{I,y}(\boldsymbol{x}_i)\right)\right]\boldsymbol{p}(\boldsymbol{x}_i) \\
\phi_{i,z}(\boldsymbol{x}_I) &= \left[\boldsymbol{p}_{,z}^T(\boldsymbol{x}_I)\boldsymbol{A}^{-1}w_{I,z} + \boldsymbol{p}^T(\boldsymbol{x}_I)\left(\boldsymbol{A}_{,z}^{-1}w_I(\boldsymbol{x}_i) + \boldsymbol{A}^{-1}w_{I,z}(\boldsymbol{x}_i)\right)\right]\boldsymbol{p}(\boldsymbol{x}_i).
\end{aligned} \tag{A.21}$$

ANNEXE B SOLVEUR IMPLICITE

On désire résoudre un système dynamique tel que

$$\boldsymbol{M}\ddot{\boldsymbol{u}} = \boldsymbol{f} \tag{B.1}$$

pour obtenir le déplacement $\boldsymbol{u}$, d'une méthode d'intégration implicite, où $\boldsymbol{M}$ est la matrice de masse, $\ddot{\boldsymbol{u}}$ le vecteur d'accélérations nodales et $\boldsymbol{f}$ le vecteur des forces nodales. Nous utilisons la méthode présentée par Baraff et Witkin [114].

On cherche les valeurs de déplacement à la fin du pas de temps :

$$\ddot{\boldsymbol{u}}(t+\Delta t) = \boldsymbol{M}^{-1}\boldsymbol{f}(t+\Delta t) \tag{B.2}$$

En utilisant le premier terme du développement en série de Taylor pour le terme de force, on obtient

$$\boldsymbol{f}(t+\Delta t) = \boldsymbol{f}(t) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Delta\boldsymbol{u} + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}, \tag{B.3}$$

avec

$$\Delta\boldsymbol{u} = \Delta t(\dot{\boldsymbol{u}}(t) + \Delta\dot{\boldsymbol{u}}) \tag{B.4}$$

$$\Delta\dot{\boldsymbol{u}} = \Delta t\ddot{\boldsymbol{u}}(t+\Delta t). \tag{B.5}$$

En remplaçant eq. B.3 dans B.2, et en utilisant B.4, on obtient

$$\ddot{\boldsymbol{u}}(t+\Delta t) = \boldsymbol{M}^{-1}\left(\boldsymbol{f}(t) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Delta\boldsymbol{u} + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}\right) \tag{B.6}$$

$$= \boldsymbol{M}^{-1}\left(\boldsymbol{f}(t) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\Delta t(\dot{\boldsymbol{u}}(t) + \Delta\dot{\boldsymbol{u}})) + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}\right) \tag{B.7}$$

$$= \boldsymbol{M}^{-1}\left(\boldsymbol{f}(t) + \Delta t\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}}(t) + \Delta t\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Delta\dot{\boldsymbol{u}} + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}\right). \tag{B.8}$$

En remplaçant le changement de vélocité $\Delta\dot{\boldsymbol{u}}$ à partir de B.5 :

$$\Delta\dot{\boldsymbol{u}} = \Delta t\boldsymbol{M}^{-1}\left(\boldsymbol{f}(t) + \Delta t\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}}(t) + \Delta t\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Delta\dot{\boldsymbol{u}} + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}\right), \tag{B.9}$$

puis, en isolant $\Delta\dot{\boldsymbol{u}}$:

$$\left(\boldsymbol{I} - \Delta t \boldsymbol{M}^{-1}\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}} - \Delta t^2 \boldsymbol{M}^{-1}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right)\Delta\dot{\boldsymbol{u}} = \Delta t \boldsymbol{M}^{-1}\boldsymbol{f}(t) + \Delta t^2 \boldsymbol{M}^{-1}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}}(t). \tag{B.10}$$

En multipliant par $\boldsymbol{M}^{-1}$ des deux côtés, on obtient un système linéaire $\boldsymbol{Ax} = \boldsymbol{b}$ à résoudre, avec la matrice d'amortissement $\boldsymbol{D}$ et la matrice de rigidité $\boldsymbol{K}$ identifiées ci-dessous :

$$\underbrace{\left(\boldsymbol{M} - \Delta t \underbrace{\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}}_{\boldsymbol{D}} - \Delta t^2 \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}}_{\boldsymbol{K}}\right)}_{\boldsymbol{A}} \underbrace{\Delta\dot{\boldsymbol{u}}}_{\boldsymbol{x}} = \underbrace{\Delta t \boldsymbol{f}(t) + \Delta t^2 \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}}_{\boldsymbol{K}} \dot{\boldsymbol{u}}(t)}_{\boldsymbol{b}}. \tag{B.11}$$

On peut ensuite résoudre le système pour trouver $\Delta\dot{\boldsymbol{u}}$, appliquer ce changement de vélocité à la valeur au temps t et calculer les prochaines positions avec B.4.

ANNEXE C RÉSOLUTION DE CONTACT

Pour résoudre les collisions entre tous les objets de la simulation, on introduit une force corrective $\boldsymbol{r}$ qui, lorsqu'elle est appliquée sur les corps impliqué dans un contact, les empêche de s'interpénétrer. Le système à résoudre (eq. 2.1) devient

$$\begin{aligned} \boldsymbol{M}\ddot{\boldsymbol{u}}(t) &= \boldsymbol{f}^{ext}(t) - \boldsymbol{K}\boldsymbol{u}(t) + \boldsymbol{r} \\ &= \boldsymbol{f}^{ext}(t) - \boldsymbol{f}^{int}(\boldsymbol{u}, \dot{\boldsymbol{u}}, t) + \boldsymbol{r}. \end{aligned} \tag{C.1}$$

En intégrant implicitement dans l'intervalle $[t_i, t_f]$, de longueur h, on obtient

$$\begin{aligned} \boldsymbol{M}(\dot{\boldsymbol{u}}_f - \dot{\boldsymbol{u}}_i) &= h(\boldsymbol{f}^{ext}(t_f) - \boldsymbol{f}^{int}(\boldsymbol{u}_f, \dot{\boldsymbol{u}}_f, t_f)) + h\boldsymbol{r}_f \\ \boldsymbol{u}_f &= \boldsymbol{u}_i + h\dot{\boldsymbol{u}}_f. \end{aligned} \tag{C.2}$$

Étant donné que $\boldsymbol{f}^{int}$ est non-linéaire, on emploie une approximation de Taylor de premier ordre

$$\boldsymbol{f}^{int}(\boldsymbol{u}_i + \Delta\boldsymbol{u}, \dot{\boldsymbol{u}}_i + \Delta\dot{\boldsymbol{u}}) = \boldsymbol{f}_i + \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{u}}\Delta\boldsymbol{u} + \frac{\partial \boldsymbol{f}^{int}}{\partial \dot{\boldsymbol{u}}}\Delta\dot{\boldsymbol{u}}. \tag{C.3}$$

En insérant cette expression dans l'équation C.1, avec $\Delta\boldsymbol{u} = \boldsymbol{u}_f - \boldsymbol{u}_i$ et $\Delta\dot{\boldsymbol{u}} = \dot{\boldsymbol{u}}_f - \dot{\boldsymbol{u}}_i$, on obtient

$$\left(\boldsymbol{M} + h\frac{\partial \boldsymbol{f}^{int}}{\partial \dot{\boldsymbol{u}}} + h^2\frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{u}}\right)\Delta\dot{\boldsymbol{u}} = -h^2\frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}}_i - h(f_i + \boldsymbol{f}^{ext}(t_f)) + h\boldsymbol{r}. \tag{C.4}$$

On définit un mouvement correctif $\Delta\dot{\boldsymbol{u}}_c$ par

$$\left(\boldsymbol{M} + h\frac{\partial \boldsymbol{f}^{int}}{\partial \dot{\boldsymbol{u}}} + h^2\frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{u}}\right)\Delta\dot{\boldsymbol{u}}_c = h\boldsymbol{r} \tag{C.5}$$

qui correspond à un déplacement

$$\Delta\boldsymbol{u} = \left(\frac{1}{h^2}\boldsymbol{M} + \frac{1}{h}\frac{\partial \boldsymbol{f}^{int}}{\partial \dot{\boldsymbol{u}}} + \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{u}}\right)^{-1}\boldsymbol{r} = \boldsymbol{C}\boldsymbol{r}, \tag{C.6}$$

où $\boldsymbol{C}$ est appelée matrice de compliance. Pour chaque point de contact α, on construit une fonction A qui relie le déplacement final $\boldsymbol{u}$ des deux objets en contact avec leur distance d'interpénétration $\boldsymbol{\delta}_\alpha$:

$$\boldsymbol{\delta}_\alpha = A_\alpha(\boldsymbol{u}_1, t_f) - A_\alpha(\boldsymbol{u}_2, t_f). \tag{C.7}$$

En définissant le jacobien de contact $H(\boldsymbol{u}) = \dfrac{\partial A}{\partial \boldsymbol{u}}$, on se retrouve au temps t à chaque contact avec

$$\Delta\boldsymbol{\delta}_\alpha = H_\alpha(\boldsymbol{u}_1)\Delta\boldsymbol{u}_1 - H_\alpha(\boldsymbol{u}_2)\Delta\boldsymbol{u}_2, \tag{C.8}$$

ce qui permet de déterminer les forces au contact lorsque celui-ci est détecté

$$\begin{aligned} \boldsymbol{r}_1 &= H_\alpha^T(\boldsymbol{u}_1)\boldsymbol{\lambda}_\alpha \\ \boldsymbol{r}_2 &= -H_\alpha^T(\boldsymbol{u}_2)\boldsymbol{\lambda}_\alpha. \end{aligned} \tag{C.9}$$

En combinant tous les contacts, on obtient la relation

$$\boldsymbol{r} = \boldsymbol{H}^T\boldsymbol{\lambda}, \tag{C.10}$$

où le vecteur $\boldsymbol{\lambda}$ contient les forces de contact à déterminer. On trouve la relation entre les distances de pénétration permises $\boldsymbol{\delta}$ et celles mesurées lors du déplacement des objets $\boldsymbol{\delta}_{free}$:

$$\boldsymbol{\delta} = \boldsymbol{HCH}^T\boldsymbol{\lambda} + \boldsymbol{\delta}_{free}. \tag{C.11}$$

On appelle opérateur Delasus le produit $\boldsymbol{W} = \boldsymbol{HCH}^T$.