Interactive Cutting of Finite Elements based Deformable Objects in Virtual Environments

Von der Fakultät für
Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
einer Doktorin der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Ing.
Lenka Jeřábková
aus Zlín, Tschechische Republik

Berichter: Prof. Christian Bischof, Ph.D.
Prof. Dr.-Ing. Matthias Teschner

Tag der mündlichen Prüfung: 14.November 2007

Abstract

There is a wide range of virtual reality (VR) applications that benefit from physically based modeling, such as assembly simulation, robotics, training and teaching (e.g., medical, military, sports) and entertainment. The dynamics of rigid bodies is well understood and several open source as well as commercial physics engines supporting articulated rigid bodies and particle systems are available. On the other hand, the simulation of deformable bodies is an objective of current research. The main application areas of deformable objects simulation in computer graphics and VR are the simulation of cloth and medical simulation. The challenge of VR applications is the real time simulation requirement. The raising computational power of the last decades allowed for adapting selected methods known from engineering sciences for interactive simulation. The simulation of cutting is especially challenging though, as most methods suffer from both performance and stability issues. Although a number of approaches have been presented over the last decade, the problem has not been solved satisfyingly, yet.

This thesis presents methods for an interactive simulation of finite elements based deformable objects as used, e.g., in VR surgical simulators. The main objectives of such simulators are stability and performance of the employed methods allowing for an interactive object manipulation including topological changes in real time. A novel method for interactive cutting of deformable objects in virtual environments is presented. The key to this method is the usage of the extended finite elements method (XFEM). The XFEM can effectively model discontinuities within an FEM mesh without creating new mesh elements and thus minimizing the impact on the performance of the simulation. The XFEM can be applied to advanced constitutive models used for the interactive simulation of large deformations. Moreover, an analysis of mass lumping techniques, showing that the stability of the simulation is guaranteed even when small portions of the material are cut is presented. The

XFEM based cutting surpasses the currently most widely used remeshing methods in both, performance and stability and is suitable for interactive VR simulation.

Further, a software architecture for physical simulation of deformable objects in VR applications is proposed. The framework is suitable for the creation of complex VR applications as, e.g., a virtual surgical trainer. It uses thread level task parallelization for the concurrent execution of visualization, collision detection, haptics and deformation. Moreover, a parallelization approach for the deformation algorithm, which is the most computationally intensive part is proposed. The presented solution based on OpenMP requires only minimal changes to the source code while achieving a speedup comparable to the results of more sophisticated approaches. The presented framework benefits from the current developments in the computing industry and allows an optimal utilization of multicore CPUs.

Zusammenfassung

Es gibt eine breite Skala von Anwendungen der virtuellen Realität (VR), die von Methoden der physikalisch basierten Modellierung profitieren können. Als Beispiele können Montagesimulation, Robotik, Training und Lehre (z.B. in der Medizin, im Militär oder im Sport) und Unterhaltung genannt werden. Die Dynamik von Festkörpern und Partikeln wurde in der Vergangenheit gut erforscht und wird zur Zeit von mehreren Open Source als auch kommerziellen Softwarepaketen unterstützt. Im Gegensatz dazu ist die Simulation deformierbarer Objekte Gegenstand aktueller Forschung. Die Hauptanwendungsgebiete für die Simulation deformierbarer Objekte in Computergraphik und VR ist die Stoff- und Kleidungssimulation sowie die medizinischen Anwendungen. Die Echtzeit-Anforderung von VR Anwendungen stellt eine große Herausforderung dar. Dank steigender Rechenleistung in den letzten Dekaden ist es möglich bestehende Methoden aus den Ingenieurswissenschaften zu übernehmen oder für interaktive Simulation zu adaptieren. Die Simulation vom Schneiden ist dennoch besonders anspruchsvoll, da die meisten Methoden zu Performanz- oder Stabilitätsproblemen führen. Obwohl in den letzten Jahren verschiedene Lösungsansätze präsentiert wurden, wurde das Problem nicht zufriedenstellend gelöst.

Diese Arbeit präsentiert Methoden für eine interaktive Simulation deformierbarer Objekte, basierend auf der Methode der finiten Elemente, die z.B. in einem virtuellen Chirurgie Simulator Verwendung finden. Die Hauptziele eines solchen Simulators sind die Stabilität und Effizienz der eingesetzten Methoden um eine interaktive Manipulation einschließlich topologischer Veränderungen in Echtzeit zu ermöglichen. In der vorliegenden Arbeit wird eine innovative Methode zum Schneiden deformierbarer Objekte in virtuellen Umgebungen präsentiert. Diese Methode basiert auf der erweiterten Methode der finiten Elemente (engl. extended finite elements method, XFEM). Mit Hilfe von XFEM können Diskontinuitäten in einem FE-Netz effizient, ohne die Erzeugung neuer Elemente, modelliert werden, wodurch der Einfluss auf

Simulationsleistung minimiert wird. Die XFEM kann mit verschiedenen Materialmodellen kombiniert und somit auch für die interaktive Simulation großer Deformationen eingesetzt werden. Des Weiteren wird die Analyse verschiedener Methoden der Diagonalisierung der Massenmatrix präsentiert und gezeigt, dass die Stabilität der Simulation unabhängig von Lage und Menge des abgeschnittenen Materials gewährleistet ist. Die XFEM basierte Methode übertrifft die zur Zeit am häufigsten eingesetzten remeshing Methoden sowohl in Effizienz als auch in der Stabilität und ist somit für interaktive VR Simulation besonders geeignet.

Weiterhin wird eine Softwarearchitektur zur Simulation deformierbarer Objekte vorgeschlagen. Das Rahmenwerk eignet sich zur Erstellung komplexer VR Anwendungen wie, z.B., eines virtuellen chirurgischen Trainers. Die nebenläufige Ausführung von Visualisierung, Kollisionserkennung, Kraftrückkopplung und Deformation wird mittels Thread-Level Parallelisierung realisiert. Außerdem wurde ein Parallelisierungsansatz für den Deformationsalgorithmus, welcher den rechenintensivsten Teil der Anwendung darstellt, entworfen und realisiert. Die präsentierte auf OpenMP basierte Lösung erfordert minimale Änderungen des Quellcodes, während gleichzeitig ein Speedup erreicht wird, der vergleichbar mit den Ergebnissen anspruchsvollerer Ansätze ist. Das vorgestellte Rahmenwerk profitiert von der gegenwärtigen Entwicklung der Computerindustrie und ermöglicht eine optimale Ausnutzung von Multicore CPUs.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die dazu beigetragen haben, dass ich mich während meiner Zeit in Aachen wohl gefühlt und mich nicht nur beruflich, sondern auch persönlich, weiterentwickelt habe.

An erster Stelle gilt mein Dank dem Leiter der VR-Gruppe am Rechen- und Kommunikationszentrum der RWTH Aachen, Dr. Torsten Kuhlen, dafür, dass er mich in seiner Gruppe aufgenommen und während der gesamten Zeit unterstützt hat. Für tatkräftige Unterstützung, fruchtbare Diskussionen und Anregungen danke ich allen meinen Kollegen, studentischen Hilfskräften und Diplomanden, insbesondere Ingo Assenmacher, Gereon Frey und Nguyen Huu Hoa. Für Einblicke in die Untiefen der Methode der finiten Elemente danke ich Jakub Jeřábek, Dr. Rostislav Chudoba und Dr. Thomas Fries. Für die Zusammenarbeit an der Parallelisierung meines Codes bedanke ich mich bei Christian Terboven und Samuel Sarholz.

Des Weiteren danke ich Prof. Christian Bischof dafür, dass er mir die Möglichkeit zur Promotion an seinem Institut eröffnete und mir die Teilnahme an zahlreichen internationalen Konferenzen ermöglicht hat. Mein herzlicher Dank gilt auch Prof. Matthias Teschner, der sich als Zweitgutachter zur Verfügung gestellt hat.

Abschließend möchte ich mich bei meinen Eltern, meinem Bruder Jakub und meinem Freund Ingo für das Vertrauen, die Geduld und die Liebe, die sie mir immer geschenkt haben, bedanken.

Contents

1 Introduction **3**
- 1.1 Related Work: VR in Medicine . . . 4
  - 1.1.1 Soft Tissue Deformation . . . 5
  - 1.1.2 Performance Optimizations . . . 7
  - 1.1.3 Cutting . . . 9
- 1.2 Contributions and Outline . . . 11

2 The Linear Elastic Material Model **13**
- 2.1 Strain . . . 14
- 2.2 Stress . . . 15
- 2.3 The Constitutive Law . . . 16
- 2.4 The Finite Elements Method . . . 17
  - 2.4.1 The Principle of Virtual Work . . . 17
  - 2.4.2 The Discretization and the Shape Functions . . . 18
  - 2.4.3 FEM Assembly . . . 19
- 2.5 Small Displacements . . . 20
- 2.6 The Corotational Method . . . 21
- 2.7 Geometrically Nonlinear FEM . . . 22
- 2.8 Dynamic FEM Simulation . . . 23
  - 2.8.1 Explicit Time Integration . . . 24
  - 2.8.2 Implicit Time Integration . . . 25

3 Cutting with the XFEM **27**
- 3.1 Modeling Discontinuities using XFEM . . . 28
  - 3.1.1 Multiple Cuts . . . 33
  - 3.1.2 The Linear XFEM . . . 34
  - 3.1.3 The Corotational XFEM . . . 36

3.1.4 The Nonlinear XFEM 38
3.2 Dynamic Simulation 39
3.3 Complexity Analysis 45

4 The Simulation Framework **49**
4.1 The Application Backbone 50
4.1.1 Mesh Generation 52
4.1.2 Multiresolution Deformation 54
4.1.3 Geometry Update during Cutting 56
4.2 Collision Detection 57
4.3 Deformation Core 59
4.4 The Simulation Loop 62

5 Shared Memory Parallelization **67**
5.1 Multicore Architectures 67
5.2 Test Cases 68
5.3 Implementation 71
5.4 Results 74

6 Conclusion **79**

CHAPTER 1

INTRODUCTION

There is a wide range of *virtual reality* (VR) applications that benefit from *physically based modeling* (PBM), such as assembly simulation, robotics, training and teaching (e.g., medical, military, sports) and entertainment. The dynamics of rigid bodies is well understood and several open source as well as commercial physics engines supporting articulated rigid bodies and particle systems are available. On the other hand, the simulation of deformable bodies is an objective of current research.

An interactive physically based simulation must be able to react on user input in real time. Moreover, the employed methods must be robust and stable under all circumstances. These requirements are crucial for an interactive simulation whereas the accuracy is sacrificed to them. The raising computational power of the last decades allowed for adapting selected methods known from engineering sciences for interactive simulation. Performance optimization techniques including adaptive multiresolution and parallelization have been proposed.

The main application areas of deformable objects simulation in computer graphics (CG) and VR are the simulation of cloth and medical simulation. From the mathematical point of view, both lead to similar systems of partial differential equations and the same numerical techniques can be used. However, medical simulation involves volumetric meshes (as opposed to surface meshes used by the cloth simulation) that lead to generally larger equation systems. Moreover, a surgery simulation requires advanced interaction techniques supported by force feedback.

Cutting is an essential manipulation task in surgery. The simulation of cutting is especially challenging though, as most methods suffer from both performance and

stability issues. Although a number of approaches have been presented over the last decade, the problem has not been solved satisfyingly yet.

In this thesis I present a novel cutting approach suitable for interactive cutting of deformable objects as used, e.g., in surgical simulation. Moreover, I introduce a parallelization approach that benefits from the current developments in the computing industry (multicore CPUs). The proposed methods have been implemented in a framework for physically based simulation of deformable objects in virtual environments. The software design of the framework is also described in this work.

The remainder of this chapter gives an overview of the related work in physically based modeling in VR and CG with special focus on medical simulation and cutting and summarizes the contributions of this thesis.

1.1 Related Work: VR in Medicine

Most of today's medical simulators only consider mechanical properties of the tissue. [Del98] points out, that in order to achieve a realistic tissue behavior, physical and physiological phenomena such as temperature, blood pressure, the relative water content, the internal organ structure, the development of pathologies and contact with other tissue have to be taken into account. For a given surgical simulation, soft tissue deformation accuracy and computation time are the two main constraints for the modeling of soft tissue. For *scientific analysis* the accuracy is far more important than the computation time. *Preoperative planning* with following intraoperative support requires a good estimation of the tissue deformation within a time frame that is acceptable for clinical use (less than 30 minutes), since several trials may be necessary [ZGHD00, AMS02]. Finally, for *surgery training*, a computation time of the order of 0.1s is required to achieve a smooth user interaction, whereas the accuracy of deformation is not of primary importance. [MHB+01] analyze the effectiveness of different user interface paradigms and system designs developed and used over the period of ten years (1991-2001) in the field of surgical planning. They also point out the difference between *surgical planning* and *surgical simulation* (surgical training). Surgical planning is an abstract process (unlike surgical simulation), the interventions are not simulated in the same difficulty and time as doing the real task. A surgical planning system should instead allow the surgeon to specify the crucial steps of the intervention as quickly and efficiently as possible. Collaboration, stereo visualization and interaction are essential.

The challenge of VR surgical trainers is in the real time simulation requirement. The first VR simulators focused on *minimal invasive surgery* (MIS). MIS minimizes the damage of healthy tissue. The relatively large cuts performed in *open surgery* are

replaced by small perforation holes, serving as entry points for customized surgical instruments and a camera (endoscope). The small extent of the tissue injury is a major gain in the patient recovery after an operation. However, the minimal invasive surgery requires special laparoscopic psychomotoric skills of the surgeon, which can be gained through an extensive training. VR is an ideal solution for this problem. Trainees can be guided through a series of tasks of progressive complexity, enabling them to develop the skills essential for good clinical practice. [LGB+06] define a taxonomy of didactic resources in laparoscopic VR in order to be able to specify requirements on a VR simulator and compare different simulators with each other or with the real world. They compare seven simulators with two scenarios using a laparoscopic box trainer and with the real operating room. [BSHW07] present a survey of VR based MIS trainers and compare more than twenty commercial MIS simulators ranging from basic-skills (navigation and hand-eye coordination) trainers to complete procedure simulators. In summary it can be said, that VR based trainers achieve a good quality of visual realism (textures, illumination) of the scene. However, a realistic and efficient simulation of soft tissue properties and high fidelity haptics of both MIS and open surgery are still subjects of ongoing scientific research.

There have been a few attempts at designing open source software toolkits meeting the challenges of medical simulation as, e.g., GiPSi [CGTS04] or SOFA [ACF+07]. Especially the authors of SOFA emphasize the need of knowledge sharing among research groups, validation of algorithms and standardization of the description of medical datasets. The following sections provide an overview of methods used in CG and VR for the simulation of deformable objects with focus on medical applications.

1.1.1 Soft Tissue Deformation

The simulation of deformable objects is an inherently interdisciplinary field combining newtonian dynamics, continuum mechanics, numerical mathematics, differential geometry, vector calculus and computer graphics. Engineering sciences have a long tradition in modeling and simulation of elastic and plastic deformations and other phenomena as crack growth and fractions. The main goal of simulation in computational mechanics is its high fidelity, whereas the computation time is not a critical factor. In interactive CG and VR applications the responsiveness of the system is crucial and is often traded for accuracy. In contrast to simulations in mechanical or civil engineering that mostly use very stiff materials as, e.g., concrete or steel, in CG applications soft materials such as cloth, hair, rubber and biological tissue are of interest. Due to the fact that the problem setting and expected simulation behavior are different, the CG can not simply adopt methods used in engineering. The challenge of interactive physically based simulation is in targeted modification

of existing approaches and finding new methods satisfying the interactivity requirement. [NMK+05] present an outstanding overview of physically based deformable models currently used in CG extending a previous survey [GM97].

The real time tissue deformation is an important constraint for medical virtual reality systems. The choice of the simulation model is influenced by computer efficiency, required accuracy and the types of manipulations that have to be performed. Several methods such as the ChainMail method [GFG+97] have been developed to perform realistic simulations of soft tissue in real time. In general, these methods are based on geometrical constraints rather than physics and thus provide less fidelity than physically based methods.

In the 1990's the intuitive *mass-spring* method was widely used for the simulation of soft objects. Mass-spring models consist of mass points linked by springs and dampers. The stiffness of the springs has to be determined experimentally. The resolution and topology of the springs is of crucial importance for the behavior of the system. Several improvements of the mass-spring method have been proposed, especially with regard to the dynamic behavior and volume preservation. Mass-spring models were used for soft tissue simulation, e.g. in [KGG96, KCM99, CK00, MBB+02]. Nowadays, the mass-spring method is used for the simulation of cloth [MTCV+04]. In medical applications, however, it is considered out-dated and has been replaced by methods based on the continuum mechanics, in particular the finite elements method (FEM). [HHR+03] compared a simplified FEM approach with the mass-spring model for surgical simulation. They show that an optimized linear FEM model requires computation time similar to the mass-spring approach, while yielding better results.

The simulation of deformable objects based on continuum mechanics was introduced to the field of CG by [TPBF87], who used it for the simulation of elastic deformations. However, due to high computational costs they only performed an off-line simulation. [Bro96] provided a detailed guide to the implementation of FEM in interactive surgery simulation. They propose the condensation and domain decomposition optimizations. [BW99] propose a band matrix optimization, a technique similar to condensation. Both are only appropriate if the FEM mesh is unchanging, such as when suturing a wound or performing just a deformation without cutting. [CDA00] compare three different approaches to soft tissue modeling based on linear elasticity theory. The first, quasi static precomputed *linear elastic model*, is computationally efficient, but does not allow for cutting. The second, called *tensor-mass* model has similar complexity as spring-mass models, but it is based on a continuous representation of the tissue. Finally, the third approach is a hybrid combination of the previous two.

The FEM-based methods used in surgical simulation in the 1990's have in common, that the stiffness matrix of the elements remains constant during the simulation.

This approach is only suitable for small deformations, as the main problem with the linear strain measure is that it is not invariant to rigid movements of the finite elements. Therefore, [MDM+02, Hau04] and [ITF04] propose similar methods treating the rigid rotation and the deformation of the elements separately. Moreover, the method proposed by [ITF04] can handle meshes with degenerated and inverted elements. The simulation of large deformations based on a nonlinear strain measure were brought back to CG applications by [OH99], who simulate fracture. In interactive medical simulation, the nonlinear FEM was used by [DDCB01] and [WDGT01].

Another group of methods based on continuum mechanics are the *meshless methods* introduced by [BYG94]. The simulated object is represented by a set of particles without a fixed neighborhood relationship. Although the meshless methods allow the simulation of deformable objects [DKS01, GQ05] including topological changes and large deformations [PKA+05, Kei06], they are computationally more expensive than the FEM and are therefore mainly used for an animation of physical phenomena, where the existence of a fixed mesh is prohibitive (e.g., melting or combination of fluids and solids) [Kei06].

1.1.2 Performance Optimizations

Interactivity and stability of the simulation are necessary conditions. The attempts to achieve a higher simulation performance can be divided into *multiresolution approaches* and *parallelization*. Most approaches in deformable modeling use a fixed space discretization. The basic idea of adaptive multiresolution approaches is to provide a high accuracy by using fine meshes at areas with large gradients in the fields of interest (e.g. stress field) and to achieve a high performance by using coarser level of detail in other regions. In a surgical simulation the regions of interest are determined by the instrument-tissue interaction and therefore they are not known a priori. [DDBC99] use a hierarchical particle representation. An octree structure is used to store the uniform space samples at each level. Similar techniques were presented by [JKWP04, NFP06]. [DDCB00] discuss a Level of Detail (LOD) technique, which uses independently defined meshes representing the quasi-uniform sampling of the simulated object at a different resolution. Different regions of the deformable body can be simulated using a different LOD. The different LOD meshes slightly overlap at their interfaces and the overlapping nodes transmit displacement information between the meshes. [WDGT01] propose a dynamic extension of the progressive mesh concept introduced by [Hop96]. The dynamic progressive meshes allow selective online refinement at the area of interest. Geometric and finite element parameters are precalculated offline for each level of the hierarchy. The update of material properties during refinement only requires local changes in the lumped stiffness matrix and consequently requires minimal computation. [WSH03], [KRS03]

and [FBD04] describe hybrid techniques that use different computational models for the area of interest and for the remaining parts. [WSH03] explicitly divide the object in an operation and a non-operation parts. The condensation technique is applied to the non-operation part. [KRS03] and [FBD04] determine the area of interest online, [KRS03] use the combination of the finite spheres method for the area of interest and the boundary elements method for the rest, [FBD04] use explicit FEM for the region close to the contact point and consider the remaining part of the object in a static configuration. [GKS02] describe an adaptive refinement approach using hierarchical base functions as opposed to a hierarchy of finite elements. The method yields equivalent adapted approximation spaces wherever the traditional mesh refinement is applicable. The main challenges of the adaptive multiresolution approaches are the automatic determination of the required resolution and an efficient refinement operation suitable for real time simulation.

Parallelization approaches using distributed memory architectures (e.g., PC-clusters) were presented by [RBST99, ZFV02, KB04, TB06]. [RBST99] describe tools and methods for partitioning and scheduling as well as design of parallel hardware and algorithms for interactive FEM computations in a surgery simulator, whereas [ZFV02, KB04, TB06] simulate cloth. The main problem is an efficient data partitioning of the irregular structures that appear in real world problems. [AR06] present a framework for coupling multiple distributed simulations in one complex VR application. Several solutions utilizing special hardware including GPUs [OLG+05, GEW05], the IBM Cell Broadband Engine [DMB+06] and the AGEIA PhysX processor [Phy06] for the performance optimization of the physical simulation of rigid or deformable objects have been proposed recently. However, in order to use the hardware acceleration, both the simulation code and data structures have to be substantially redesigned in order to map to the specific hardware, which is a nontrivial task requiring special and deep knowledge of the hardware architecture used. Moreover, the end users have to purchase a specific hardware in order to be able to use the optimizations. Another promising strategy is the employment of general purpose multicore architectures [OH05, SL05] as, e.g., the AMD Opteron or the Intel Xeon dualcore processors allowing for parallel processing of multiple tasks. [TPB07] present a parallelization approach for cloth simulation on an AMD Opteron machine with two dualcore processors. They use the same parallelization techniques (domain decomposition followed by a matrix restructuring) as on a PC cluster [TB06]. They designed an own multithreaded parallel programming model. The OpenMP API [Boa05] provides a declarative shared model parallelization model for the C/C++ and Fortran programming languages, while permitting portability. The OpenMP directives extend the programming languages with SIMD constructs, work-sharing constructs, and synchronization constructs, and they provide support for the sharing and privatization of data. Compared to lower-level parallelization approaches as, e.g., Posix-Threads, OpenMP requires the least design changes of an existing serial code while achieving competitive results [JKT+07].

1.1.3 Cutting

This section presents a short overview of the previous FEM-based approaches related to the simulation of cutting, cracks and fracture in CG and VR. [BSMM02] presents a survey of interactive cutting techniques in virtual surgery until 2002. The described techniques are categorized according to how the solutions address the following issues: definition of the cut path, primitive removal and remeshing, number of new primitives created, when remeshing is performed and the representation of the cutting tool. A state of the art cutting simulation enables an interactive movement of the cutting tool through the mesh and thus defining the cutting path in a natural way. The cut is created and visualized immediately or with a short delay while the tool is moving (progressive cutting) as opposed to waiting until the tool no longer intersects the tissue. The cutting tool is represented by a single point, single edge or triangle, allowing for simple intersection tests. At the same time, a more complex model is used for rendering. In the following, the most representative approaches are summarized.

The main difficulty a cutting simulator has to deal with is the primitive remeshing. Here, the visual primitives (triangles) have to be distinguished from the simulation primitives (e.g., springs or finite elements). Both kinds of primitives are arranged in meshes, that have to be cut in a consistent way. However, to avoid consistency problems and interpolation between both meshes, many authors use the boundary of the (volumetric) simulation mesh for visualization. The most methods for surgical cutting published so far require the FEM elements to be aligned with the cut. This can be achieved either by constraining the cut to the borders of existing elements, or by splitting the elements along the cut, or by snapping of the elements' borders to the cut, or by combination of these methods.

[CDA00] *remove* the tetrahedra touched by the cutting tool thus forming a gap in the mesh. No new elements are created, therefore the stability of the simulation is not harmed. However, the method violates the mass conservation law and the created cut contains unpleasing visual artifacts. A number of authors use the *tetrahedra subdivision method*. [BMG99, VHML99, GCMS00] subdivide the tetrahedra according to predefined templates. [BGTG03] presents a state machine that tracks the topology of the intersections in the tetrahedral mesh to ensure consistent remeshing of neighboring tetrahedra. The original tetrahedron is replaced by up to seventeen new tetrahedra. [MK00] generate a minimal set of new elements to replace the cut tetrahedron. The main drawback of this group of methods is the creation of small ill-shaped elements (*slivers*) causing numerical instability of the simulation. To avoid the drawback of the previous two methods, [NvdS01] snap the nodes of the existing elements to the trajectory of the cut. Although no new elements are created, this method can still lead to degenerated elements. They are detected and removed. Moreover, if the snapping distance is large, the parameters

of the mesh have to be updated. [SHGS06] use a combination of snapping and subdivision. However, they only support nonprogressive cutting. [WBG07] use an approach based on arbitrary convex finite elements. They remesh the FE mesh after a cut, however, the newly created elements are not subdivided into tetrahedra, thus avoiding the potential creation of many ill-conditioned elements. However, slivers cannot be avoided completely.

As for the approaches used in the computer animation of cracks and fracture, [OH99] constrains the beginning of the cut to go through an existing node and uses tetrahedra subdivision or snapping during the crack propagation. [MBF04] introduced the *virtual node algorithm*. In contrast to the previous methods, the crack does not have to be aligned to the elements' boundary. Instead of subdividing an element, one or more replicas of the cracked element are created. The graphical representation of the material within the original element is fragmented, and the portions are assigned to the particular replicas. In order to avoid instabilities, each element replica duplicates the material of the original element instead of splitting it along the crack, which would be physically correct, but would lead to instabilities in case of material slivers. In the original virtual nodes algorithm the number of cuts per element is limited in that each fragment has to contain at least one node of the original FE mesh. These limitations were resolved by [SDF07], thus allowing for arbitrary cuts of tetrahedral elements. However, [SDF07] do not mention how the masses are updated. All, [OH99], [MBF04] and [SDF07] consider an offline simulation rather than an interactive cutting or breaking of objects.

Meshless methods avoid the existence of an explicit mesh. However, in case of topological changes, additional structures are required to identify topologically separated particles. Moreover, a nontrivial dynamic resampling has to be performed in the vicinity of newly created incisions [PKA+05, SOG06]. In all above approaches the cut or crack is represented by a piecewise linear path.

The approach presented in this thesis is based on the *extended finite elements method* (XFEM) as proposed by [BB99]. The XFEM method introduces a local enrichment in subregions with discontinuities. The enrichment is based on the *partition of unity method* (PUM) [MB96, BM98]. The XFEM method was first developed for two-dimensional linear elastic fracture mechanics in civil engineering. A single crack was considered. Subsequently, it has been extended to many applications, such as material interfaces in solids and fluids [BZXC03, SCMB01, CBM03], crack growth [MDB99], or arbitrary branched and intersecting cracks and holes [DMD+00]. In classical structural mechanics the focus is laid on the accuracy of the simulation rather than the speed of computation. Moreover, only very stiff materials (such as concrete) undergoing small deformations in a static or quasi static setup were considered [SP03]. [VVW04] were the first to use the XFEM in a medical context. However, only a static simulation in 2D (MRI images) and small deformations were considered. The simulated cut consisted of a single line segment. The suitability of

XFEM for a dynamic simulation in 3D and especially the interactive cutting of soft objects as used, e.g., in virtual surgery simulation was not analyzed before. Moreover, in all previous publications the XFEM is used in the context of linear FEM, which is only suitable for small deformations. The XFEM can effectively model discontinuity regions within an FEM mesh. An important advantage of the XFEM compared to remeshing based methods is its accuracy and the reduced requirements on the simulation mesh. As no new elements are created during a cut, the impact on performance of the simulation is minimized [JJCK06, JJCK07]. Moreover, this method is very well suited to the structure of existing finite element codes. The remeshing based methods suffer from stability problems when small parts of the tissue are separated. The reasons for the instability are twofold. First, ill-shaped elements are created, and second, the masses of the new elements can be very low. In the XFEM no ill-shaped elements are created. The choice of an appropriate enrichment and a mass lumping technique is the key to a stable dynamic simulation using XFEM.

1.2 Contributions and Outline

This thesis presents methods for an interactive simulation of finite elements based deformable objects as used, e.g., in VR surgical simulators. The main objectives of such simulators are stability and performance of the employed methods allowing for an interactive object manipulation including topological changes in real time. The main contributions are:

- **simulation of interactive cutting without remeshing**. I prove the suitability of XFEM for interactive cutting of deformable objects in virtual environments. Furthermore, I discuss the stability issues of the dynamic simulation and analyze different mass lumping approaches in order to guarantee the stability of the simulation regardless of the location of the cut. Finally, I analyze the impact of cutting with this approach on the simulation performance. The XFEM approach is described in chapter 3.

- **the combination of XFEM and large deformations**. I show how XFEM can be applied to the corotational and geometrically nonlinear constitutive models, which are the most commonly used methods for the simulation of large deformations in CG. This is described in chapter 3 as well.

- **framework for physically based simulation of deformable objects**. I propose a software architecture that structures the simulation layer of interactive VR applications, e.g., in the field of surgical simulation. As a proof of concept, I implemented the proposed methods and integrated them into a

software framework. Based on this framework, I developed a prototype of an open surgery simulator as well as other testing applications. The structure and main features of this framework are discussed in chapter 4.

- **parallelization on multicore CPUs**. I present a parallelization approach for the above mentioned framework that benefits from the current developments in the computing industry (chip level parallelism). The proposed solution based on OpenMP requires only minimal changes to the source code while achieving a significant speedup of the simulation on commodity hardware. The parallelization approach is described in detail in chapter 5.

Chapter 2 is an overview of the principles of continuum mechanics, FEM and numerical mathematics needed for a better understanding of the remainder sections.

CHAPTER 2

THE LINEAR ELASTIC MATERIAL MODEL

The linear elastic model is the best-known of the elastic models. Systems derived from the linear elastic model are easy to solve compared to other elastic models and are therefore used in interactive simulations even though the linearity assumption is only valid for infinitesimal displacements of real materials. Robert Hooke first formulated linear elastic behavior in 1676. He observed that under small loads and deformations, deflection of many materials is linearly proportional to the applied load (Hooke's Law). For a linear spring the Hooke's law reads

$$f = ku, \tag{2.1}$$

where f is the applied force load, k is the spring constant and u is the spring elongation.

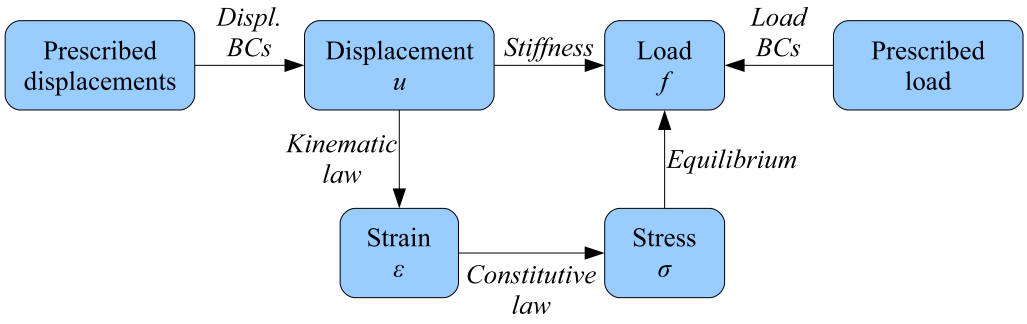


Figure 2.1: Tonti diagram for the external and internal mechanical quantities.

For more complicated cases than a linear spring the stiffness equation cannot be expressed in such a simple way. Therefore, internal quantities are introduced to the

mathematical model. The behavior of materials can be described by the relationship between the four physical fields: displacement, strain, stress and force (cp. Fig. 2.1). The displacement, which corresponds to the movement of particles in the elastic continuum, is linked to the internal displacement measure (strain) by kinematic compatibility conditions. The internal force measure (stress) is linked to strain by constitutive (material) laws, and finally, the stress is linked to the external load by the equilibrium equations. Forces and displacements can be seen as external factors that can be observed and measured. Moreover, forces and displacements can be prescribed by defining *boundary conditions* (BC), corresponding, e.g., to fixing of an object part or an application of an interaction force. Strain and stress are internal mathematical tools to measure the effects of displacements and forces respectively. The relationship between stress and strain is what determines the actual physical behavior of the continuum. Different continuum models are used to describe different material behavior (elasticity, plasticity, viscosity). For more details on the continuum mechanics see, e.g., [Mal69].

2.1 Strain

Strain is a measure of a body's deformation. For a one dimensional spring, strain can be defined as the ratio of elongation with respect to the original length. For a 3D body, strain can be expressed as a matrix.

$$\epsilon = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \tag{2.2}$$

with

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \sum_{k=1}^{3} \frac{\partial u_k}{\partial x_i} \cdot \frac{\partial u_k}{\partial x_j}\right) \tag{2.3}$$

called the Green's strain tensor. u is the displacement vector, x is a coordinate, and the indices i, j range over the three coordinates in three dimensional space. The Green's strain tensor is an objective measure of the deformation, i.e. it is invariant under rigid body translations and rotations. The nonlinear equation (2.3) has to be used for deformations in which the displacement gradients are large. For small deformations, the linearized Cauchy's tensor is often used.

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.4}$$

The nonlinearity of the equation 2.3 is called geometrical nonlinearity as opposed to material nonlinearity meaning a nonlinear stress-strain relation.

2.2 Stress

Stress is a measure of the inner forces within a solid that balance a given set of external forces and body forces. The stress or traction vector is a force vector per unit area. By defining a set of internal planes perpendicular to the Cartesian coordinate axes the stress state at an arbitrary internal point can be described relative to x, y, and z coordinate directions. Stress can be expressed in terms of three traction vectors acting on three perpendicular planes. Each of these vectors can be decomposed into three mutually orthogonal components. The subscript notation used for the nine stress components have the following meaning: σ_{ij} - stress on the i-th plane along the j-th direction (Fig. 2.2). Nine stress components from

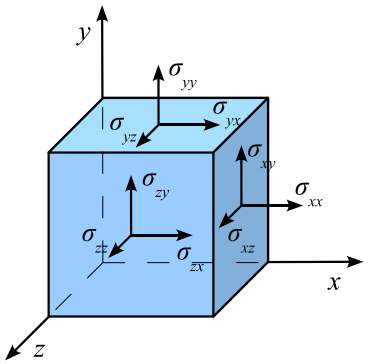


Figure 2.2: Stress components at an internal point represented by in an infinitesimal cube.

three planes are needed to describe the stress state. These nine components can be organized into a matrix.

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \tag{2.5}$$

This grouping of the nine stress components is known as the stress tensor. The diagonal components are normal to the reference surfaces and represent the normal stress. The off-diagonal components are tangential to the reference surfaces and represent shear stresses. Normal stress tends to change the volume of the material whereas the shear stress tends to deform the material without changing its volume. The off-diagonal shear stresses are symmetrical ($\sigma_{xy} = \sigma_{yx}$, $\sigma_{xz} = \sigma_{zx}$, $\sigma_{zy} = \sigma_{yz}$).

The equilibrium equations have the form

$$f_i = \sum_{j=1}^{3} \frac{\partial \sigma_{ji}}{\partial x_j} \tag{2.6}$$

which is equavalent to

$$\mathbf{f} = \nabla \cdot \sigma \tag{2.7}$$

where ∇ is the divergence operator ($\nabla \cdot \mathbf{A} = \frac{\partial \mathbf{A}_x}{\partial x} + \frac{\partial \mathbf{A}_y}{\partial y} + \frac{\partial \mathbf{A}_z}{\partial z}$).

2.3 The Constitutive Law

The constant of proportionality between applied tensile stress and resulting strain for linear elastic materials was defined by Thomas Young and is known as the Young modulus of elasticity.

$$E = \frac{\sigma_{tensile}}{\epsilon_{tensile}} \tag{2.8}$$

When a linear elastic material is loaded axially in tension in the xx direction the material contracts laterally in the yy and the zz directions. The ratio between the lateral and axial deformation is known as the Poisson's ratio ν and it's range is $[0, 0.5]$.

$$\nu = -\frac{\epsilon_{lateral}}{\epsilon_{axial}} \tag{2.9}$$

For isotropic materials loaded axially in the xx direction:

$$\nu = -\frac{\epsilon_{yy}}{\epsilon_{xx}} = -\frac{\epsilon_{zz}}{\epsilon_{xx}} \tag{2.10}$$

The Lamé material constants (λ, μ) are sometimes used instead of (E, ν) for 3-D analysis of isotropic linear elastic materials because the elastic constitutive relations can be written more concisely. The drawback to the Lamé material constants is that they cannot be as easily measured as (E, ν).

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{2.11}$$

$$\mu = \frac{E}{2(1+\nu)} \tag{2.12}$$

The relation between stress and strain is given by a constitutive equation. For elastic materials, the local stress depends only on the local instantaneous value of the strain. In general, the stress-strain relation can be expressed by a fourth order tensor of elastic constants, which has $3^4 = 81$ components. However, it can be shown that for an anisotropic material only 21 components are independent. For an

isotropic material only two constants are needed. These are the Lamé constants λ and μ. The constitutive law then reads

$$\sigma = \lambda tr(\epsilon)\mathbb{I} + 2\mu\epsilon \tag{2.13}$$

where $tr()$ denotes the trace of a matrix (i.e., the sum of all diagonal elements of a matrix) and $\mathbb{I}$ is an $[3 \times 3]$ identity matrix. Equation (2.13) is often written in a matrix form, where ϵ and σ are rearranged as a 6-component vectors and $\mathbb{C}$ is a material matrix, also known as rigidity, or constitutive matrix, cp. equation (2.14).

$$\sigma = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{yz} \\ \tau_{zx} \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{yz} \\ \gamma_{zx} \\ \gamma_{xy} \end{bmatrix} = \mathbb{C}\epsilon \tag{2.14}$$

$$\gamma_{yz} = \epsilon_{yz} + \epsilon_{zy} = 2\epsilon_{yz} \quad \gamma_{zx} = \epsilon_{zx} + \epsilon_{xz} = 2\epsilon_{zx} \quad \gamma_{xy} = \epsilon_{xy} + \epsilon_{yx} = 2\epsilon_{xy}$$
$$\epsilon_x = \epsilon_{xx} \quad \epsilon_y = \epsilon_{yy} \quad \epsilon_z = \epsilon_{zz}$$

2.4 The Finite Elements Method

The Finite Elements Method is a numerical technique for solving boundary value problems. In its application, the object or system is represented by a geometrically similar model consisting of multiple, linked, simplified representations of discrete regions called finite elements (FE). Laws of continuum mechanics are applied to each element, and a system of simultaneous equations is constructed. The system of equations is solved for unknown values using the techniques of linear algebra or nonlinear numerical schemes, as appropriate. While being an approximate method, the accuracy of the FEM method can be improved by refining the mesh in the model using more elements and nodes. This chapter presents a strongly simplified view of the FEM, which is sufficient to understand this thesis. More details can be found, e.g., in [Bat86], [ZTZ05] and numerous other FEM books.

2.4.1 The Principle of Virtual Work

The equilibrium equation (2.7) together with the boundary conditions is called *strong form* of the boundary value problem. The FEM is based on a different formulation

called the *weak form* or the *principle of virtual work* or the *principle of virtual displacement*, which is mathematically a variational method. According to the principle of virtual work, for a body in equilibrium the work done by internal stresses cancel out, and the internal virtual work done reduces to the work done by the applied external forces.

$$\tilde{W}_I = \tilde{W}_E. \tag{2.15}$$

The external virtual work of the force f passing through the virtual node displacement $\tilde{u}$ is

$$\tilde{W}_E = \tilde{u}^T f. \tag{2.16}$$

The virtual node displacements $\tilde{u}$ cause virtual strain $\tilde{\epsilon}$. The internal virtual work associated with this load is

$$\tilde{W}_I = \int_V \tilde{\epsilon}^T \sigma dV. \tag{2.17}$$

When the strain-displacement equations are expressed as $\epsilon = \mathbb{B}u$, then after inserting equations (2.16, 2.17, 2.14), equation (2.15) leads to

$$\tilde{u}^T f = \tilde{u}^T \left(\int_V \mathbb{B}^T \mathbb{C}\mathbb{B} \, \mathrm{d}V \right) u \tag{2.18}$$

and thus the governing equation of linear elasticity becomes

$$f = \left(\int_V \mathbb{B}^T \mathbb{C}\mathbb{B} \, \mathrm{d}V \right) u \tag{2.19}$$

where the term $\int_V \mathbb{B}^T \mathbb{C}\mathbb{B} \, \mathrm{d}V$ corresponds to stiffness.

2.4.2 The Discretization and the Shape Functions

In the FEM, the continuum is approximated using a mesh of finite elements (e.g., triangles, tetrahedra or hexahedra) connected at nodes. The forces and displacements are only evaluated at the nodes. Any external forces have to be expressed as node forces with an equivalent impact. The element internal forces and displacements are interpolated from the nodal forces and displacements using a set of functions, typically referred to as *shape*, *basis* or *interpolation* functions. The displacement of an arbitrary point can be computed as

$$u(x) = \sum_{i=1}^{n} \Phi_i(x) \mathbf{u}_i \tag{2.20}$$

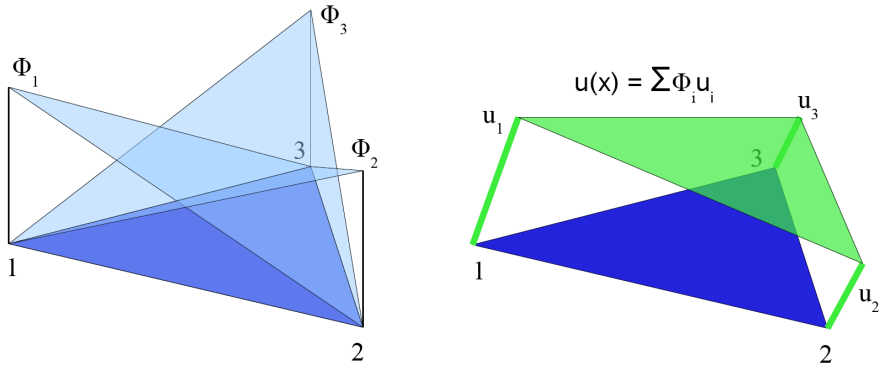


Figure 2.3: In the FEM, the nodal displacements $\mathbf{u}_i$ are interpolated using the shape functions Φ_i in order to get a continuous displacement $u(x)$ within an element.

where n is the number of element nodes, Φ_i are the element shape functions and $\mathbf{u}_i$ are the nodal displacements. The shape functions fulfill the Kronecker delta property, meaning that the value of Φ_i is 1 in the node i and 0 in all other nodes. Consequently, $u(x_i) = \mathbf{u}_i$. For a linear triangle and tetrahedron, the shape functions are identical to the barycentric coordinates (cp. Fig. 2.3).

Equation (2.19) can be written in terms of discrete vectors $\mathbf{f}$ and $\mathbf{u}$.

$$\mathbf{f}_i = \sum_{j=1}^{n} \int_V \mathbb{B}_i^T \mathbb{C} \mathbb{B}_j \, \mathrm{d}V \, \mathbf{u}_j = \sum_{j=1}^{n} \mathbb{K}_{ij} \mathbf{u}_j \tag{2.21}$$

$\mathbb{K}$ is a stiffness matrix. For 3D mechanical problems, the $\mathbb{K}$ matrix of a finite element is a positive definite symmetrical matrix with the size of $[3n \times 3n]$, where n is the number of element nodes.

2.4.3 FEM Assembly

In order to obtain a deformation of the whole system, the elements have to be put together while satisfying the conditions of displacement compatibility and equilibrium [ZTZ05]. The condition of displacement compatibility means that the displacement of all elements meeting at the same node must be the same, whereas according to the equilibrium condition, the sum of forces exerted by all elements meeting at a node must balance the external force at that node. If a single node a is considered, the elements' forces contributions can be expressed as

$$\sum_i \mathbf{f}_a^i = \sum_i \mathbb{K}_a^i \mathbf{u} = \mathbf{f}_a \tag{2.22}$$

where the summation only concerns elements contributing to node a and $\mathbb{K}_a^i$ is the a-th row of the stiffness matrix of the i-th element. The displacement vector $\mathbf{u}$ is common to all elements. The global stiffness matrix can be thus assembled by simply adding the contributions of the element stiffness matrices while considering the global node numbering. This assembly process is a fundamental feature of the direct stiffness method. The resulting global stiffness matrix is a sparse symmetric positive definite matrix of the size $[3n \times 3n]$, where n is the number of nodes in the whole system and the sparsity pattern corresponds to the elements' connectivity.

2.5 Small Displacements

When the simulated displacement gradients are small, the linear Cauchy's strain (cp. equation (2.4)) can be used. The strain-displacement matrix $\mathbb{B}$ is defined as

$$\begin{bmatrix} \mathbb{B}_1 & \dots & \mathbb{B}_n \end{bmatrix} \tag{2.23}$$

where $\mathbb{B}_i$ are matrices containing the partial derivations of the shape functions (cp. equations (2.4) and (2.20)).

$$\mathbb{B}_i = \begin{bmatrix} \frac{\partial \Phi_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial \Phi_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial \Phi_i}{\partial z} \\ \frac{\partial \Phi_i}{\partial y} & \frac{\partial \Phi_i}{\partial x} & 0 \\ \frac{\partial \Phi_i}{\partial z} & 0 & \frac{\partial \Phi_i}{\partial x} \\ 0 & \frac{\partial \Phi_i}{\partial z} & \frac{\partial \Phi_i}{\partial y} \end{bmatrix} \tag{2.24}$$

The $[3 \times 3]$ components of the element stiffness matrix are defined as

$$\mathbb{K}_{ij} = \int_V \mathbb{B}_i^T \mathbb{C} \mathbb{B}_j \, \mathrm{d}V \tag{2.25}$$

where i and j are the indices of the element nodes. For a linear tetrahedron (using linear shape functions) with linear elastic material both the strain-displacement matrix $\mathbb{B}$ and the material matrix $\mathbb{C}$ are constant. Once the object discretization has been determined, the element stiffness matrix $\mathbb{K}$ can be computed.

The linearization of the Green's strain (cp. equation (2.3)) known as the Cauchy's strain (cp. equation (2.4)) results in a constant stiffness matrix, that has to be initialized once at the beginning of the simulation as opposed to being recomputed in each

simulation step. The better computational efficiency is traded for accuracy, which becomes noticeable when large deformations occur. In particular, the Cauchy's strain tensor is not invariant to rigid body rotations. It produces ghost forces, which cause the deformed object to blow up unnaturally. Therefore, more advanced methods have to be used for the simulation of large displacements. The corotational method treats the rigid rotation and the deformation of the elements separately while using the linear Cauchy's strain. Alternatively, the nonlinear Green's strain can be used. Fig. 2.4 shows the comparison of the three methods.

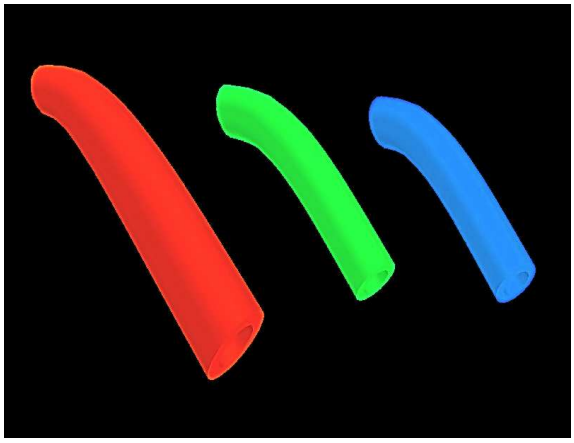


Figure 2.4: A tube bended under gravity with the left side fixed simulated using the linear (red), corotational (green) and geometrically nonlinear (blue) methods.

2.6 The Corotational Method

The corotational method was introduced to the CG community by [MDM+02] and was improved by [HS04] and [MG04]. It is based on the linear FEM with Cauchy's strain, however, the elements are aligned with their reference configuration prior to the force computation. The deformation forces computed for the aligned configuration are then rotated back to the current configuration. Thus, the deformation forces are computed as

$$\mathbf{f}_i = \mathbb{R} \sum_{j=1}^{n} \mathbb{K}_{ij} \left(\mathbb{R}^T \mathbf{p}_j - \mathbf{p}_{0j} \right) \tag{2.26}$$

where $\mathbb{R}$ is the rotation matrix needed for the alignment of the current and initial configurations, $\mathbb{K}_{ij}$ is the corresponding submatrix of the linear stiffness matrix defined in equation (2.25), $\mathbf{p}$ are the positions of the element nodes in the current deformed configuration and $\mathbf{p}_0$ are the initial positions of the element nodes. Note that $\mathbf{p} = \mathbf{p}_0 + \mathbf{u}$.

[MDM+02] and [HS04] propose different techniques for the estimation of the rotation matrix. The time efficiency and robustness of this step strongly influence the quality of the simulation. [MDM+02] determines an orthonormal base in both, the initial and the deformed configurations. The transformation matrix between these two bases contains the sought rotation matrix. The translational part of the transformation is omitted and as the bases are orthonormal, the transformation matrix does not contain any scaling or shearing factors. As an alternative to this approach, [HS04] uses polar decomposition of the deformation gradient.

2.7 Geometrically Nonlinear FEM

The nonlinear FEM can be derived with the help of the deformation gradient and Piola-Kirchhoff stress tensors rather than in terms of a stiffness matrix. For a finite element, the deformation gradient $\mathbb{F}$ can be expressed using the shape functions and the nodal displacements.

$$\mathbb{F} = \frac{\partial u}{\partial x} + \mathbb{I} = \sum_{i=1}^{n} \frac{\partial \Phi_i}{\partial x} \mathbf{u}_i + \mathbb{I} = \sum_{i=1}^{n} \beta_i \mathbf{u}_i + \mathbb{I} \tag{2.27}$$

where $\beta_i = \frac{\partial \Phi_i}{\partial x}$ and $\mathbb{I}$ is an identity matrix. In a three dimensional space, the deformation gradient is a $[3 \times 3]$ matrix as both u and x are 3D vectors. The Green's strain tensor can be expressed as (cp. equation (2.3))

$$\epsilon = \frac{1}{2} \left(\mathbb{F}^T \mathbb{F} - \mathbb{I} \right) \tag{2.28}$$

The second Piola-Kirchhoff stress tensor is defined as

$$\mathbb{S}_{ij} = \lambda \sum_{k=1}^{3} \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij} \tag{2.29}$$

where λ and μ are Lamé material constants and δ_{ij} is Kronecker delta. The deformation force can be computed as (cp. equation (2.7))

$$\mathbf{f} = \int_V \nabla \cdot \left(\mathbb{S}\mathbb{F}^T \right) \, dV \quad \text{or} \quad \mathbf{f}_i = \beta_i \int_V \left(\mathbb{S}\mathbb{F}^T \right) \, dV \tag{2.30}$$

where the term $\mathbb{S}\mathbb{F}^T$ corresponds to the first Piola-Kirchhoff stress tensor. For a linear element, the deformation gradient and the derived strain and stress are constant over the element as the partial derivatives of the shape functions are constant.

2.8 Dynamic FEM Simulation

The deformation of the object is given by the displacement of the nodes according to the acting external and internal forces. In an interactive simulation the applied forces change in time and the virtual objects have to react to them in real time. Therefore, the FEM has to be simulated dynamically. Mass and damping factors are added to the static deformation forces (cp. equations (2.21, 2.26, 2.30)), in order to account for inertia and energy dissipation. The dynamic deformation is described by the following formula.

$$\mathbb{M}\ddot{\mathbf{u}} + \mathbb{D}\dot{\mathbf{u}} + \mathbb{K}\mathbf{u} = \mathbf{f}_{ext} \tag{2.31}$$

$\mathbb{M}$ is the mass matrix, $\mathbb{D}$ is the damping matrix, $\mathbb{K}$ is the global stiffness matrix, $\mathbf{u}$ is the vector of nodal displacements and $\mathbf{f}_{ext}$ is the external load. For simplicity the notation of a linear FEM is used here. In case of the corotational or nonlinear methods, the term $\mathbb{K}\mathbf{u}$ is replaced by the expression for the respective deformation forces (cp. equations (2.26, 2.30)).

Equation (2.31) is an initial boundary value problem. Finding its solutions means finding such a displacement vector $\mathbf{u}$, that the external forces are balanced by the internal forces. The acceleration of the elements' nodes is given by the difference of the external load and the internal forces consisting of the deformation forces and damping.

$$\mathbf{a} = \ddot{\mathbf{u}} = \mathbb{M}^{-1}\left(\mathbf{f}_{ext} - \mathbb{D}\dot{\mathbf{u}} - \mathbb{K}\mathbf{u}\right) = \mathbb{M}^{-1}\mathbf{F} \tag{2.32}$$

If the external forces are balanced by the body's internal forces, the resulting force $\mathbf{F}$ acting on the body is zero and consequently, the body acceleration is zero. If the forces are not balanced, the body or its parts undergo nonzero acceleration. The consistent mass matrix is defined as

$$\mathbb{M}_{ij} = \rho \int_V \Phi_i \Phi_j \, dV \tag{2.33}$$

where ρ is the material density. In oder to evaluate equation (2.32) the inverse of the mass matrix is required. Therefore the mass matrix is usually diagonalized using a technique called *mass lumping*. The most widely used form of a lumped mass matrix is the row summation

$$\bar{\mathbb{M}}_{ii} = \rho \sum_j \int_V \Phi_i \Phi_j \, dV = \rho \int_V \Phi_i \, dV \tag{2.34}$$

as the sum of all Φ_j is one.

For the damping matrix, the Rayleigh formula (2.35) is mostly used [Bat86].

$$\mathbb{D}_{ii} = \alpha \mathbb{M}_{ii} + \beta \mathbb{K}_{ii} \tag{2.35}$$

α and β are scaling factors.

The acceleration of the nodes caused by the unbalanced external and internal body forces is a key to the object deformation. The nodal acceleration can be integrated in time to obtain the velocities of the nodes, which can be integrated in time again to obtain the nodal displacements. The velocity is defined as

$$\mathbf{v} = \dot{\mathbf{u}} \tag{2.36}$$

In order to perform the numerical integration of the acceleration and velocity, the simulation time t is discretized into time steps Δt. The acceleration at a given time t can be evaluated using equation (2.32). The initial conditions at the beginning of the simulation ($t = 0$), are set to zero nodal displacements $\mathbf{u}(0) = 0$ and zero velocities $\mathbf{v}(0) = 0$. Solving the equation (2.31) numerically means finding the velocities and displacements in time steps $t = \Delta t, t = 2\Delta t, \ldots$ Theoretically, any numerical method for solving the initial value problem of ordinary differential equations (ODE) can be used. Practically, for interactive applications, the applied method has to be fast enough to enable for real time simulation.

2.8.1 Explicit Time Integration

Explicit methods are generally easy to implement, as in each step only the information from previous steps is used [PFTV92]. When applied to the equation (2.31), the explicit Euler method leads to

$$\Delta\mathbf{u} = \Delta t \cdot \mathbf{v}_t \tag{2.37}$$

$$\Delta\mathbf{v} = \Delta t \cdot \mathbf{a}_t \tag{2.38}$$

The new displacement is obtained using the current velocity. The current acceleration is evaluated using the current displacement and velocity, then it is used to obtain the new velocity.

The main drawback of explicit methods is, that they are limited by a critical simulation time step, above which they become unstable. The critical time step of the simulated system is given by

$$\Delta t_c = \frac{2}{\omega_{max}} \tag{2.39}$$

where ω_{max} is the maximum eigenfrequency, also called natural vibration frequency, of the system. The eigenfrequencies can be determined as eigenvalues of the following problem

$$\mathbb{K}\mathbf{x} = \omega^2\mathbb{M}\mathbf{x} \tag{2.40}$$

where ω^2 are the eigenvalues, ω are the eigenfrequencies and $\mathbf{x}$ are the corresponding eigenmodes (eigenvectors). For a real time simulation, Δt must be larger than the time needed to compute the new values of acceleration, velocity and displacement. Therefore, if Δt becomes too small (for the sake of stability), even a method with a simple single step can become too computationally expensive for real time applications.

2.8.2 Implicit Time Integration

Implicit methods are theoretically unconditionally stable, but they require the evaluation of partial derivatives of the deformation forces and the solution of a large, possibly nonlinear, system of equations [PFTV92]. The implicit Euler method leads to

$$\Delta \mathbf{u} = \Delta t \cdot \mathbf{v}_{t+\Delta t} \tag{2.41}$$

$$\Delta \mathbf{v} = \Delta t \cdot \mathbf{a}_{t+\Delta t} \tag{2.42}$$

Equation (2.42) uses the acceleration at time $t + \Delta t$ to obtain the velocity at time $t+\Delta t$. However, the velocity and displacement at time $t+\Delta t$ are needed to evaluate the acceleration at time $t + \Delta t$. The Newton-Raphson method can be used to solve the system of equations. Let $\mathbf{b}_u$ and $\mathbf{b}_v$ be

$$\mathbf{b}_u = \Delta \mathbf{u} - \Delta t \cdot \mathbf{v}_{t+\Delta t} \tag{2.43}$$

$$\mathbf{b}_v = \mathbb{M} \cdot \Delta \mathbf{v} - \Delta t \cdot \mathbf{F}_{t+\Delta t} \tag{2.44}$$

The equations are solved if $\mathbf{b}_u$ and $\mathbf{b}_v$ are equal to zero. The unknowns are $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$. The Newton-Raphson method evaluates $\mathbf{b}_u$ and $\mathbf{b}_v$ for some starting values of $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$. Then it approximates $\mathbf{b}_u$ and $\mathbf{b}_v$ with the derivatives at the starting point and computes the values of $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$ as the point where the derivative becomes zero. The new estimates of $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$ are used in the next iteration. The difference between the new and old values of $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$ is the iteration step $\mathbf{s}_u$ and $\mathbf{s}_v$ respectively.

The Newton-Raphson method iterates until the iteration step is below a given threshold. The main drawback of the Newton-Raphson method is the need for the derivative of the forces. If the derivative changes in time, it has to be computed in each simulation time step, which is quite time consuming. However, if the stiffness matrix is constant, the force derivative is constant as well and has to be computed only once at the beginning of the simulation.

CHAPTER 3

CUTTING WITH THE XFEM

An interactive cutting simulation is an essential feature of a surgery trainer. However, the interactive progressive cutting of a deformable FEM mesh is a challenging problem. The simulation of irreversible destructive phenomena such as cracks or fracture plays an important role in mechanical and civil engineering. The simulation can complement or replace experiments with rare, expensive or unsafe materials (e.g., explosives). The goal is to allow an arbitrary element dissection without suffering from the restrictions imposed by small sliver elements.

The number and quality of the FEM elements have a direct impact on the simulation performance and stability. From the technical point of view, the simulation of a surgical cut and the simulation of crack or fracture are similar and have to deal with the same stability issues. Although a number of different approaches have been presented recently, the problems have not been solved satisfyingly. In this chapter I prove the suitability of XFEM for interactive cutting of deformable objects in virtual environments. I will show how XFEM can be applied to the linear, corotational and geometrically nonlinear constitutive models, which are the most commonly used methods for the simulation of deformations in CG. Then the stability issues of a dynamic simulation are discussed and different mass lumping approaches are analyzed in order to guarantee the stability of the simulation regardless of the location of the cut. Finally the impact of cutting on the simulation performance is analyzed.

3.1 Modeling Discontinuities using XFEM

In the FEM the displacement within an element is interpolated using equation (2.20). When a discontinuity (e.g., a cut or crack) has to be added, the surrounding mesh nodes are enriched by an additional global discontinuous enrichment function multiplied by shape functions with a local support, thereby leading to a local discontinuous enrichment (cp. Fig. 2.3 and 3.1). The corresponding number of nodal degrees of freedom (DOF) is added.

$$u(x) = \sum_{i=1}^{n} \Phi_i(x)\mathbf{u}_i + \sum_{j=1}^{n} \Phi_j^*(x)\psi_j(x)\mathbf{a}_j \tag{3.1}$$

where $\psi(x)_j$ are the discontinuous enrichment functions and $\mathbf{a}_j$ are the added nodal DOF. The shape functions of the added DOF $\Phi_j^*(x)$ are not necessarily identical to the shape functions of the corresponding nodes. For example, it is possible to use higher order shape functions $\Phi_i(x)$ together with linear enrichment shape functions $\Phi_j^*(x)$. In the following, $\Phi_j^*(x) = \Phi_i(x)$ is considered. The XFEM is

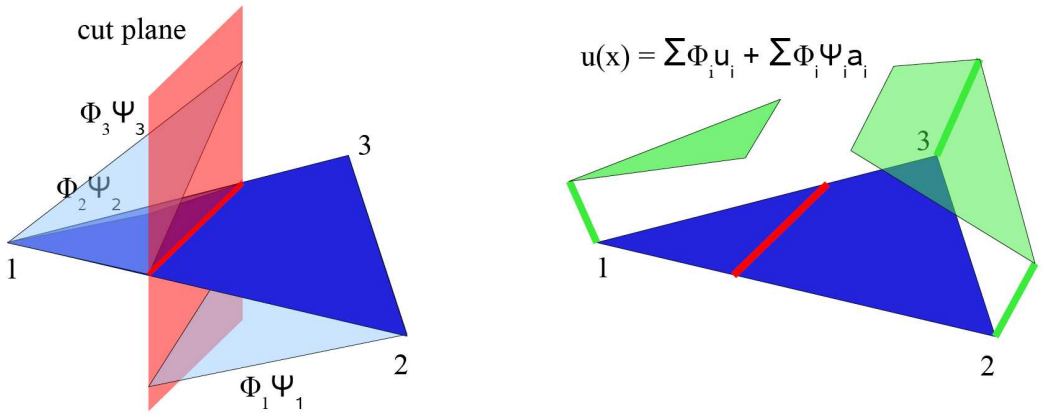


Figure 3.1: The discontinuous enrichment functions Ψ_i lead to a discontinuous displacement field $u(x)$.

based on the partition of unity concept [MB96, BM98]. The functions $\Phi_i^*(x)$ build a partition of unity in local parts of the domain. The enrichment function can be any arbitrary discontinuous function provided that it is discontinuous over the crack or cut domain. The simplest and the most widely used $\psi(x)$ is a generalized Heaviside function [BB99] also known as the $sign()$ function

$$\Psi(x) = H(x) = \begin{cases} +1 & \text{above the crack} \\ -1 & \text{below the crack} \end{cases} \tag{3.2}$$

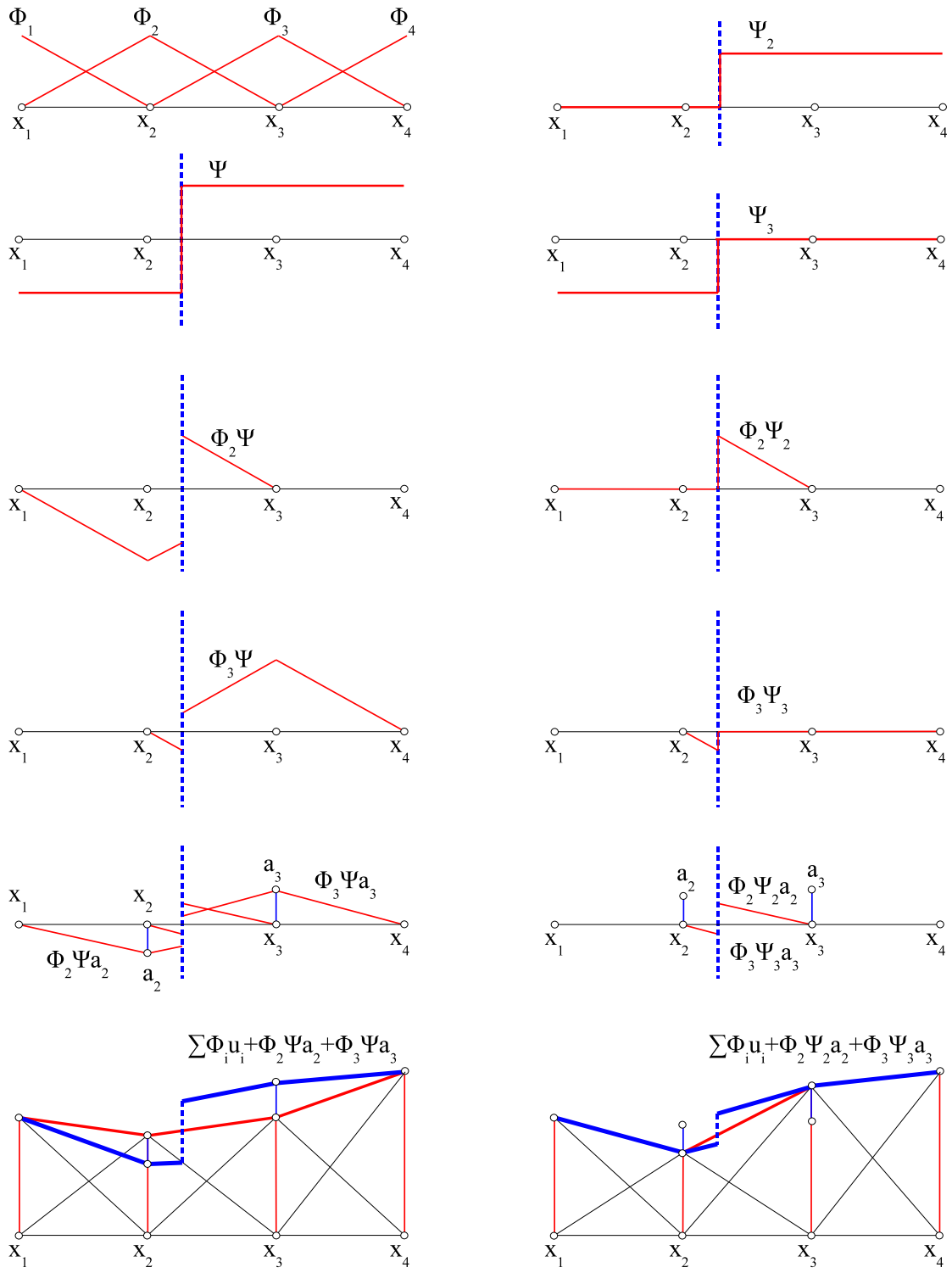


Figure 3.2: The comparison of two different enrichment approaches. On the left, the enrichment function $\psi(x) = H(x)$ is used, whereas on the right, the shifted enrichment function $\psi_i(x) = \frac{1}{2}\left(H(x) - H_i\right)$ is used.

The enrichment of the nodes impacts all elements sharing the enriched nodes, i.e., the one ring of neighbors of the element containing the cut. Due to the additional enrichment term of equation (3.1) the shape functions together with the local enrichment functions $\Phi_i^*(x)\psi_i(x)$ do not have the Kronecker delta property in general (cp. to chapter 2.4.2). Consequently, the displacement of the enriched nodes has to be computed as a sum of the components $\mathbf{u}_i + \Psi_i\mathbf{a}_i$. This fact also complicates the treatment of boundary conditions. *Shifted* enrichment functions, that are zero at all nodes, can be used instead [ZB03].

$$\psi_i(x) = \frac{H(x) - H_i}{2} \tag{3.3}$$

where H_i is the value of $H(x)$ at the i-th node. The division by 2 ensures the Kronecker delta property. The effect of shifting is that the enrichment contributions only appear within the element that has been cut and vanish at its border and outside the element (cp. Fig. 3.2), which leads to an enormous simplification of the implementation.

Figure 3.3 shows how the choice of the enrichment function influences the physical meaning of the original and the added DOF. When the generalized Heaviside function is used for enrichment, the $\mathbf{u}_i$ loose their physical meaning of nodal displacement. On the other hand, when the shifted enrichment is used, the nodal displacements of non-enriched and enriched nodes are stored directly in $\mathbf{u}_i$, the contributions of $\mathbf{a}_i$ are only needed to determine the displacement within an enriched element.

The added enrichment functions and nodal DOF in fact double the element and allow for the decoupling of the dissected parts. The same effect can be reached by the replacement of the original continuous shape functions by discontinuous functions (cp. Fig. 3.4) corresponding to a linear combination of the original shape functions and the XFEM enrichment. For an enrichment using the generalized Heaviside function, the substitution is defined as

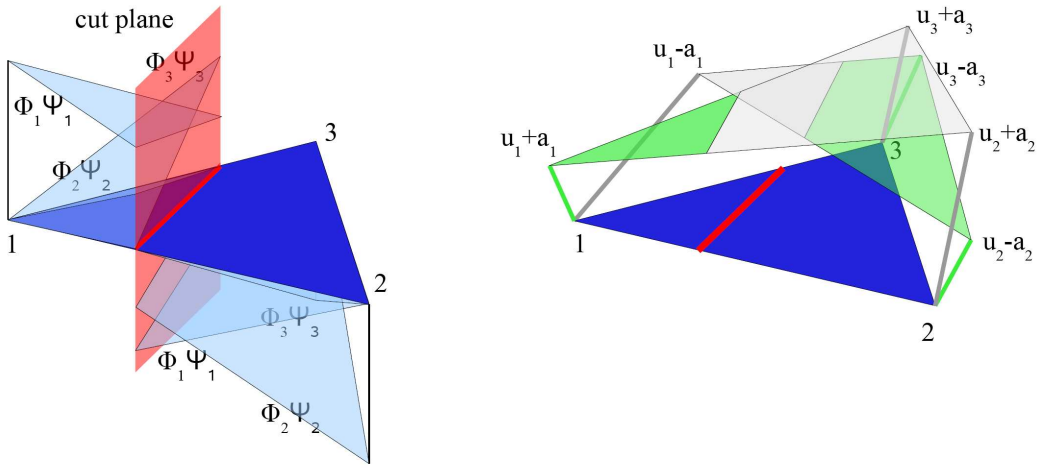
$$\Phi_i'(x) = \frac{1}{2}\left(\Phi_i(x) + \Phi_i(x)\Psi_i(x)\right) \tag{3.4}$$

$$\Phi_i''(x) = -\frac{1}{2}\left(\Phi_i(x) - \Phi_i(x)\Psi_i(x)\right) \tag{3.5}$$

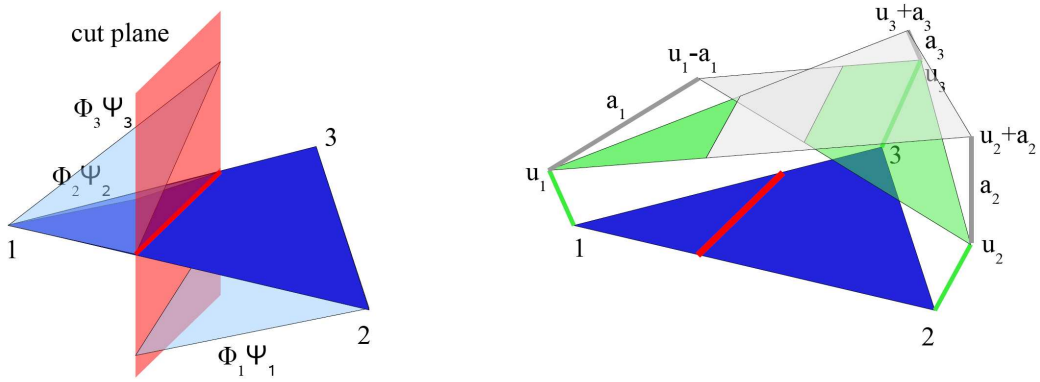
The scale by $\pm\frac{1}{2}$ ensures the Kronecker delta property. The effect is, that $\Phi_i' = \Phi_i$ and $\Phi_i'' = 0$ above the cut and $\Phi_i' = 0$ and $\Phi_i'' = \Phi_i$ below the cut. The corresponding nodal DOF $\mathbf{u'}_i$ and $\mathbf{u''}_i$ have the physical meaning of displacement and substitute the original DOF as

$$\mathbf{u'}_i = \mathbf{u}_i + \mathbf{a}_i \tag{3.6}$$

$$\mathbf{u''}_i = \mathbf{u}_i - \mathbf{a}_i \tag{3.7}$$



(a) generalized Heaviside function $\psi(x) = H(x)$



(b) shifted enrichment function $\psi_i(x) = H(x) - H_i$

Figure 3.3: Physical meaning of the nodal DOF depending on the enrichment function used.

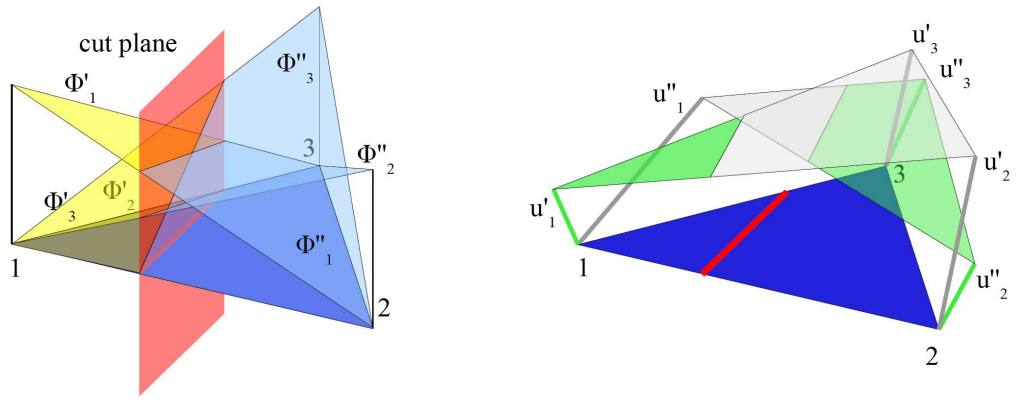


Figure 3.4: DOF substitution.

For a linear tetrahedron this case is identical to the virtual node algorithm [MBF04], where the virtual nodes correspond to the DOF in each element's part without material, i.e., nodes $\mathbf{u''}_1, \mathbf{u'}_2, \mathbf{u'}_3$ in Figure 3.4. A similar duality of additional or substitutional shape functions can be found in the context of multiresolution techniques [GKS02, KGS03], where it is called true hierarchical and quasi-hierarchical refinement.

It has to be noted, that all above mentioned versions of XFEM enrichment are physically equivalent, i.e., they will all lead to the same displacements. However, as already mentioned, the choice of the enrichment influences the 'straight-forwardness' of the implementation, and, as will be shown later (cp. chapter 3.2), also the numerical stability of the simulation.

The principle of enrichment functions is general and thus can be used with any type of finite elements such as shells, volumetric elements and even springs and any type of constitutive model. It allows not only the simulation of *strong discontinuities*, such as cuts or cracks, where parts of material are separated, but also the simulation of *weak discontinuities*, such as material interfaces in solids and fluids or the interface between solid and fluid. In a weak discontinuity, the displacements are continuous, however, they contain a kink, and thus the displacement gradient is discontinuous. For more details on the simulation of weak discontinuities using XFEM, I refer to [BZXC03, SCMB01, CBM03]. In this thesis, strong discontinuities are considered. Although the XFEM principles throughout this thesis are demonstrated on a single cut per element, the extension to multiple cuts is straight forward and will be described in the following section.

3.1.1 Multiple Cuts

When an element is cut through, two independent components exist, and the number of DOF with corresponding shape functions is doubled. Each of the components can be cut again, creating a cut branch (cp. Fig. 3.5). Every new cut leads to a new independent component and a new set of DOF and enrichment functions. An important principle is that the support of the enrichment covers a previously existing element component and not necessarily the whole element. In the situation

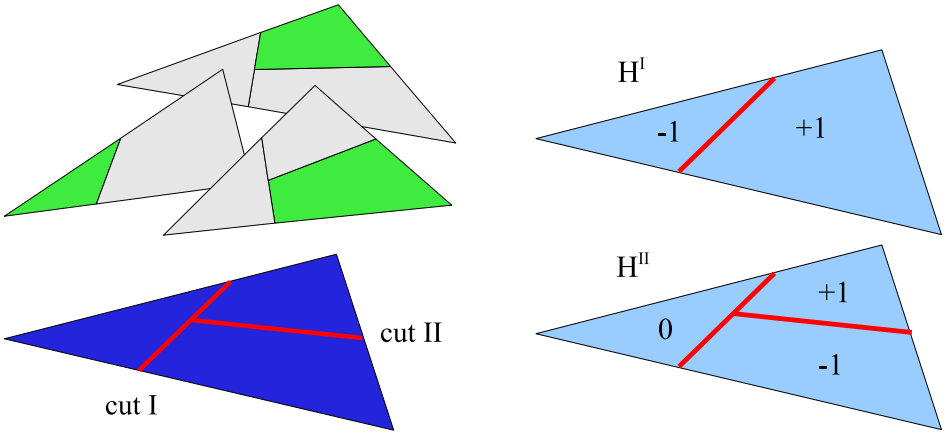


Figure 3.5: Multiple cuts per element. Each new cut leads to a new independent component.

of Fig. 3.5, cut I has been created first, followed by cut II. Cut I was enriched using the function H^I, whereas the enrichment function H^{II} corresponds to cut II. At the end, three independent components exist, with 9 nodal DOF in total. The displacement within an element can be computed as follows

$$\begin{aligned} u(x) &= \sum_{i=1}^{n} \Phi_i(x)\mathbf{u}_i + \sum_{j=1}^{n} \Phi_j^*(x)\psi_j^I(x)\mathbf{a}_j^I + \sum_{j=1}^{n} \Phi_j^*(x)\psi_j^{II}(x)\mathbf{a}_j^{II} \\ &= \sum_{i=1}^{n} \Phi_i(x)\mathbf{u}_i + \sum_{j=1}^{m} \Phi_j^*(x)\psi_j(x)\mathbf{a}_j \end{aligned} \tag{3.8}$$

where m is the number of added DOF. Equation (3.8) is a generalization of equation (3.1), where j runs over all added DOF. The enrichment functions $\Psi_j(x)$ are considered to be global as are the functions H^I and H^{II}. The local nodal enrichment is realized by the multiplication of $\Psi_j(x)$ by $\Phi_j^*(x)$ with local support. Note that function H^{II} is zero on the left side of cut I. Thus, the DOF added with cut II will not influence the left side of cut I.

For a more complex crack branching, a hierarchical structure of the components as shown in Fig. 3.6 is useful. The group nodes contain the enrichment data of a given

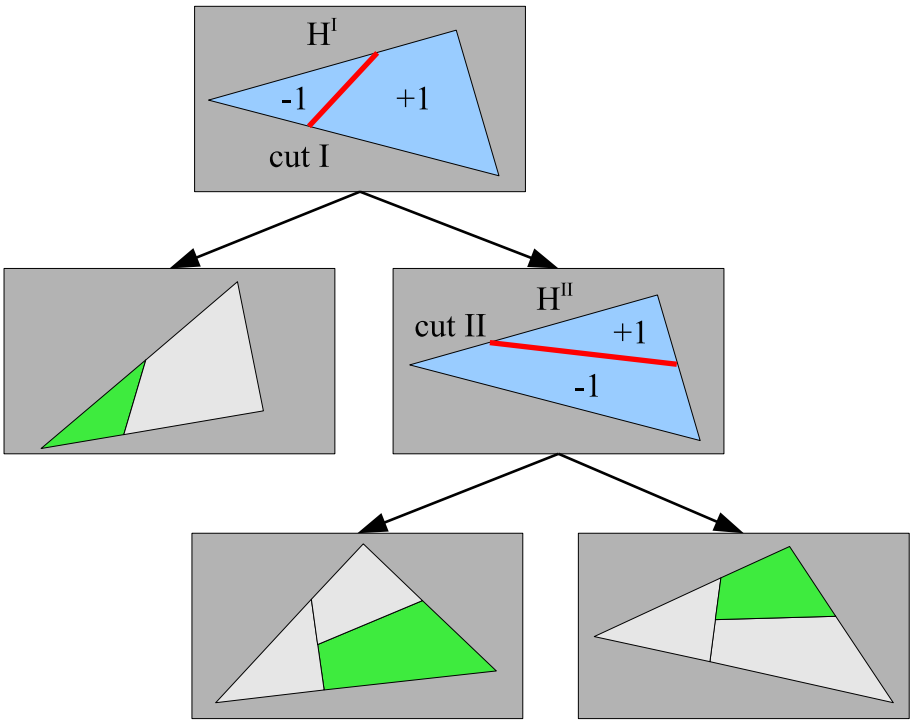


Figure 3.6: Multiple cuts per element - a hierarchical cut structure

cut, whereas the leaf nodes represent the resulting independent components. The area (volume) of each component is defined by a sequence of cuts from the root node of the hierarchy to the corresponding leaf.

The following sections describe the enrichment of three most commonly used constitutive models in CG: a physically and geometrically linear element based on Cauchy's strain, a corotational element and St. Venant-Kirchhoff element based on Green's strain.

3.1.2 The Linear XFEM

When a discontinuity has to be added, new DOF are appended to the displacement and force vectors $\mathbf{u}$ and $\mathbf{f}$ respectively.

$$\mathbf{u}^X = \begin{bmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_n & \mathbf{a}_1 & \dots & \mathbf{a}_n \end{bmatrix}^T \tag{3.9}$$

$$\mathbf{f}^X = \begin{bmatrix} \mathbf{f}_1 & \dots & \mathbf{f}_n & \mathbf{f}_1^a & \dots & \mathbf{f}_n^a \end{bmatrix}^T \tag{3.10}$$

The the components of the enriched stiffness matrix are computed as (cp. equation (2.25)).

$$\mathbb{K}_{ij}^{X} = \int_{V} \mathbb{B}_{i}^{XT} \mathbb{C} \mathbb{B}_{j}^{X} \, dV \tag{3.11}$$

The enriched strain matrix has the form

$$\mathbb{B}^{X} = \left[\begin{array}{cccccc} \mathbb{B}_1 & \dots & \mathbb{B}_n & \psi_1 \mathbb{B}_1 & \dots & \psi_n \mathbb{B}_n \end{array} \right] \tag{3.12}$$

The enrichment function $\psi_i(x)$ takes on constant values over the domain above (V_a) and below (V_b) the cut plane and only changes value from one to another. Thus, the volume integral in equation (3.11) can be split into two parts

$$\mathbb{K}_{ij}^{X} = \underbrace{\int_{V_a} \mathbb{B}_{i}^{XT} \mathbb{C} \mathbb{B}_{j}^{X} \, dV}_{above} + \underbrace{\int_{V_b} \mathbb{B}_{i}^{XT} \mathbb{C} \mathbb{B}_{j}^{X} \, dV}_{below} \tag{3.13}$$

The enriched stiffness matrix can be divided into four parts

$$\mathbb{K}^{X} = \left[\begin{array}{cc} \mathbb{K}^{uu} & \mathbb{K}^{ua} \\ \mathbb{K}^{au} & \mathbb{K}^{aa} \end{array} \right] \tag{3.14}$$

where $\mathbb{K}^{uu}$ corresponds to the original DOF, whereas $\mathbb{K}^{ua}$, $\mathbb{K}^{au}$ and $\mathbb{K}^{aa}$ correspond to the added DOF. After inserting equation (3.12) into equation (3.13) and assuming constant matrix $\mathbb{B}$, the stiffness matrix components become

$$\mathbb{K}_{ij}^{uu} = \mathbb{K}_{ij} \tag{3.15}$$

$$\mathbb{K}_{ij}^{ua} = \left(\frac{V_a}{V} \Psi_{aj} + \frac{V_b}{V} \Psi_{bj} \right) \mathbb{K}_{ij} \tag{3.16}$$

$$\mathbb{K}_{ij}^{au} = \left(\frac{V_a}{V} \Psi_{ai} + \frac{V_b}{V} \Psi_{bi} \right) \mathbb{K}_{ij} \tag{3.17}$$

$$\mathbb{K}_{ij}^{aa} = \left(\frac{V_a}{V} \Psi_{ai} \Psi_{aj} + \frac{V_b}{V} \Psi_{bi} \Psi_{bj} \right) \mathbb{K}_{ij} \tag{3.18}$$

$\mathbb{K}^{uu}$ is identical to the original stiffness matrix of the non-enriched element, whereas for $\mathbb{K}^{ua}$, $\mathbb{K}^{au}$ and $\mathbb{K}^{aa}$ the original stiffness matrix is multiplied by constant factors depending on the size of the dissected volumes and the cut side of the given node (above or below). Ψ_{ai} denotes the value of the function $\Psi_i(x)$ above the cut plane, whereas Ψ_{bi} is its value below the cut plane. When the shifted enrichment is used

(cp. equation (3.3)), the components of the stiffness matrix result in

$$\mathbb{K}_{ij}^{uu} = \mathbb{K}_{ij} \tag{3.19}$$

$$\mathbb{K}_{ij}^{ua} = \begin{cases} -\frac{V_b}{V}\,\mathbb{K}_{ij} & \text{if } H_j = +1 \\ \frac{V_a}{V}\,\mathbb{K}_{ij} & \text{if } H_j = -1 \end{cases} \tag{3.20}$$

$$\mathbb{K}_{ij}^{au} = \begin{cases} -\frac{V_b}{V}\,\mathbb{K}_{ij} & \text{if } H_i = +1 \\ \frac{V_a}{V}\,\mathbb{K}_{ij} & \text{if } H_i = -1 \end{cases} \tag{3.21}$$

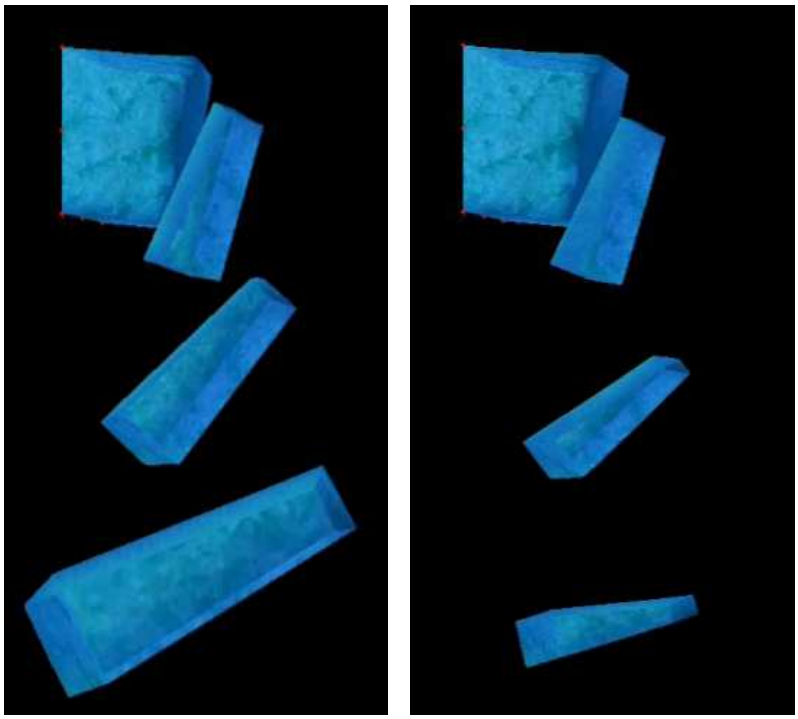
$$\mathbb{K}_{ij}^{aa} = \begin{cases} \frac{V_b}{V}\,\mathbb{K}_{ij} & \text{if } H_i = H_j = +1 \\ \frac{V_a}{V}\,\mathbb{K}_{ij} & \text{if } H_i = H_j = -1 \\ 0 & \text{if } H_i \neq H_j \end{cases} \tag{3.22}$$

The linear FEM is the most simple FE model as the stiffness matrix remains constant during the simulation. However, it is well known that the linear FEM is only suitable for the simulation of small displacements. In particular, the underlying Cauchy's strain measure is not rotationally invariant and leads to disturbing artifacts when the elements are rotated. Fig. 3.7 illustrates this phenomenon. It shows three simulation time steps of a slice that has been cut of a cube and is falling down under gravity and rotating at the same time. In the simulation on the left, the Cauchy's strain has been used, whereas the the Green's strain has been used for the simulation on the right. The linear XFEM might be sufficient for the simulation of cracks or partial cuts in a stiff material, when the created opening is not very wide. In cases when the dissected parts undergo a large deformation or the simulated object falls apart, advanced methods have to be used.

3.1.3 The Corotational XFEM

When the element is cut, the aligning rotation is different for the dissected parts (cp. Fig. 3.8). Thus, the deformation forces in an enriched element are computed as follows (cp. equations (2.26) and (3.13))

$$\mathbf{f}_i^X = \sum_{j=1}^{n} \underbrace{\mathbb{R}_a \int_{V_a} \mathbb{B}_i^{XT} \mathbb{C} \mathbb{B}_j^X \, dV \; \left(\mathbb{R}_a^T \mathbf{p}_j^X - \mathbf{p}_{0j}^X\right)}_{above} + \sum_{j=1}^{n} \underbrace{\mathbb{R}_b \int_{V_b} \mathbb{B}_i^{XT} \mathbb{C} \mathbb{B}_j^X \, dV \; \left(\mathbb{R}_b^T \mathbf{p}_j^X - \mathbf{p}_{0j}^X\right)}_{below} \tag{3.23}$$



(a) Cauchy's strain. (b) Green's strain.

Figure 3.7: Three simulation time steps of a falling dissected slice. Cauchy's strain measure leads to disturbing artifacts when the object is rotated.

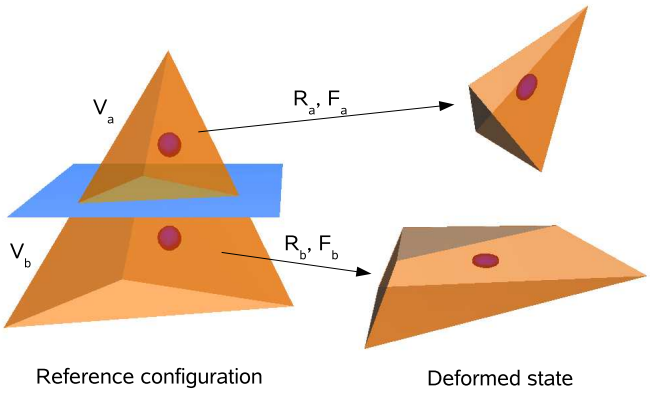


Figure 3.8: The dissected parts V_a, V_b of an element undergo generally different rotations R_a, R_b and deformations F_a, F_b. The displacement of an arbitrary point within an element can be determined using equation (3.1). Especially, all points that are above or below the cut in the (non-deformed) reference configuration will remain on the same side of the cut in the deformed state.

where $\mathbb{R}_a$ and $\mathbb{R}_b$ are the rotations of the element parts above and below the cut respectively. Note that $\mathbf{p}^X = \mathbf{p}_0^X + \mathbf{u}^X$. After inserting equation (3.2) into equation (3.23), the deformation force corresponding to a standard DOF ($0 \leq i \leq n$) becomes

$$\mathbf{f}_i = \frac{V_a}{V}\mathbb{R}_a \sum_{j=1}^{n} \mathbb{K}_{ij} \left(\mathbb{R}_a^T \mathbf{p}_{aj} - \mathbf{p}_{0j}\right) + \frac{V_b}{V}\mathbb{R}_b \sum_{j=1}^{n} \mathbb{K}_{ij} \left(\mathbb{R}_b^T \mathbf{p}_{bj} - \mathbf{p}_{0j}\right) \tag{3.24}$$

where

$$\mathbf{p}_{aj} = \mathbf{p}_{0j} + \mathbf{u}_j + \Psi_{aj}\mathbf{a}_j \tag{3.25}$$

$$\mathbf{p}_{bj} = \mathbf{p}_{0j} + \mathbf{u}_j + \Psi_{bj}\mathbf{a}_j \tag{3.26}$$

For the added DOF, equation (3.23) results in

$$\mathbf{f}_i^a = \frac{V_a}{V}\mathbb{R}_a\Psi_{ai} \sum_{j=1}^{n} \mathbb{K}_{ij} \left(\mathbb{R}_a^T \mathbf{p}_{aj} - \mathbf{p}_{0j}\right) + \frac{V_b}{V}\mathbb{R}_b\Psi_{bi} \sum_{j=1}^{n} \mathbb{K}_{ij} \left(\mathbb{R}_b^T \mathbf{p}_{bj} - \mathbf{p}_{0j}\right) \tag{3.27}$$

If the shifted enrichment function (cp. equation (3.3)) is used, equations (3.25) - (3.27) result in

$$\mathbf{p}_{aj} = \begin{cases} \mathbf{p}_{0j} + \mathbf{u}_j & \text{if } H_j = +1 \\ \mathbf{p}_{0j} + \mathbf{u}_j + \mathbf{a}_j & \text{if } H_j = -1 \end{cases} \tag{3.28}$$

$$\mathbf{p}_{bj} = \begin{cases} \mathbf{p}_{0j} + \mathbf{u}_j - \mathbf{a}_j & \text{if } H_j = +1 \\ \mathbf{p}_{0j} + \mathbf{u}_j & \text{if } H_j = -1 \end{cases} \tag{3.29}$$

$$\mathbf{f}_i^a = \sum_{j=1}^{n} \begin{cases} -\frac{V_b}{V}\mathbb{R}_b\mathbb{K}_{ij} \left(\mathbb{R}_b^T \mathbf{p}_{bj} - \mathbf{p}_{0j}\right) & \text{if } H_i = +1 \\ \frac{V_a}{V}\mathbb{R}_a\mathbb{K}_{ij} \left(\mathbb{R}_a^T \mathbf{p}_{aj} - \mathbf{p}_{0j}\right) & \text{if } H_i = -1 \end{cases} \tag{3.30}$$

3.1.4 The Nonlinear XFEM

When the element is cut, the deformation gradient, strain and stress have to be determined for each part separately (cp. Fig. 3.8). The deformation force can then be computed as (cp. equation (2.30))

$$\mathbf{f}_i^X = \underbrace{\int_{V_a} \beta_i^X \mathbb{S}_a \mathbb{F}_a^T \, dV}_{above} + \underbrace{\int_{V_b} \beta_i^X \mathbb{S}_b \mathbb{F}_b^T \, dV}_{below} \tag{3.31}$$

where $\mathbb{F}_a$, $\mathbb{S}_a$, $\mathbb{F}_b$, $\mathbb{S}_b$ are the deformation gradient and the second Piola-Kirchhoff stress above and below the cut respectively, and β^X is defined as

$$\beta^X = \left[\begin{array}{cccccc} \beta_1 & \dots & \beta_n & \psi_1\beta_1 & \dots & \psi_n\beta_n \end{array} \right] \tag{3.32}$$

with $\beta_i = \frac{\partial \Phi_i}{\partial x}$. Note, that β^X depends on the discontinuous enrichment functions and is therefore discontinuous as well. However, it is constant over the domains above and below the cut respectively. The discontinuous deformation gradient of an enriched element can be computed as (cp. equation(2.27))

$$\mathbb{F} = \frac{\partial u}{\partial x} + \mathbb{I} = \sum_{i=1}^{n} \beta_i \mathbf{u}_i + \sum_{i=1}^{n} \Psi_i \beta_i \mathbf{a}_i + \mathbb{I} = \beta^X \cdot \mathbf{u}^X + \mathbb{I} \tag{3.33}$$

where $\beta^X \cdot \mathbf{u}^X$ is a dot product of the vectors β^X and $\mathbf{u}^X$. The corresponding second Piola-Kirchhoff stress tensor can be computed using equations (2.3) and (2.29).

For the deformation force corresponding to a standard DOF ($0 \leq i \leq n$), equation (3.31) results in

$$\mathbf{f}_i = \beta_i \left(V_a \mathbb{S}_a \mathbb{F}_a^T + V_b \mathbb{S}_b \mathbb{F}_b^T \right) \tag{3.34}$$

whereas for the DOF appended after the cut we get

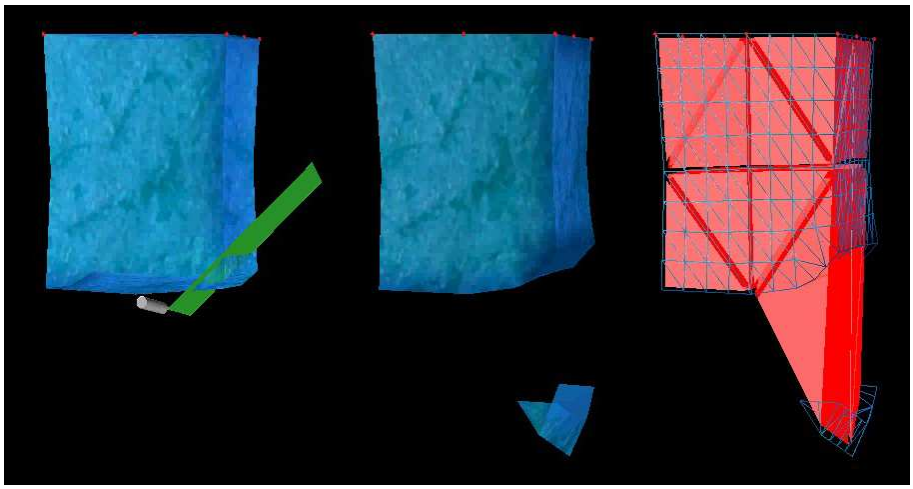
$$\mathbf{f}_i^a = \beta_i \left(\Psi_{ai} V_a \mathbb{S}_a \mathbb{F}_a^T + \Psi_{bi} V_b \mathbb{S}_b \mathbb{F}_b^T \right) \tag{3.35}$$

If the shifted enrichment function (cp. equation (3.3)) is used, equation (3.35) results in

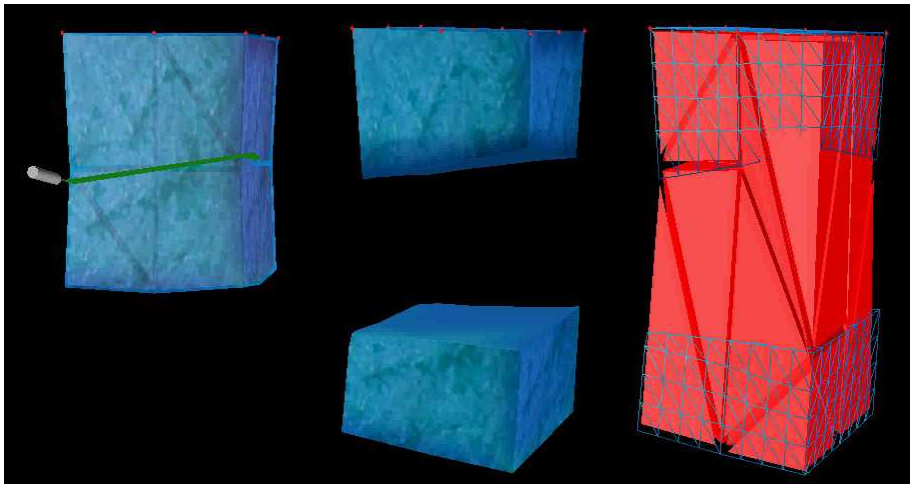
$$\mathbf{f}_i^a = \begin{cases} -\beta_i V_b \mathbb{S}_b \mathbb{F}_b^T & \text{if } H_i = +1 \\ \beta_i V_a \mathbb{S}_a \mathbb{F}_a^T & \text{if } H_i = -1 \end{cases} \tag{3.36}$$

3.2 Dynamic Simulation

Sliver elements as shown, e.g., in Fig. 3.9, have negative impact on the numerical accuracy and stability of both, explicit and implicit methods. The stability of the simulation can be influenced by the choice of the enrichment function and a mass lumping technique. Depending on the mass matrix and the position of the cut, the ω_{max} given by equation (2.40) can become very high (possibly infinite), pushing the critical timestep Δt_c towards zero. Generally, if $\mathbb{M}$ and $\mathbb{K}$ are positive definite, all ω^2 are real and positive and thus all eigenfrequencies are positive. The non enriched stiffness matrix is symmetric positive definite and the non enriched mass matrix is diagonal with positive values and thus also symmetric positive definite. In this



(a) Dissection of a small volume.



(b) Approximately 50% of finite elements are cut in a way that V_a or V_b are nearly zero.

Figure 3.9: Dissections of a cube leading to small material slivers. The upper face of a deformable cube is fixed, the dissected volume is falling down under gravity (left and middle). The underlying finite elements (usually invisible to the user) are visualized on the right. The enriched elements are stretching between both dissected parts.

section I will analyze the stability criteria for an enriched element using the shifted enrichment function (cp. equation (3.3)) depending on mass lumping techniques.

Similarly to the enriched stiffness matrix, the enriched mass matrix has the form

$$\mathbb{M}^X = \begin{bmatrix} \mathbb{M}^{uu} & \mathbb{M}^{ua} \\ \mathbb{M}^{au} & \mathbb{M}^{aa} \end{bmatrix} \tag{3.37}$$

The components of the consistent mass matrix of an enriched element are defined as

$$\mathbb{M}_{ij}^{uu} = \rho \int_V \Phi_i \Phi_j \, dV \tag{3.38}$$

$$\mathbb{M}_{ij}^{ua} = \rho \int_V \Phi_i \Phi_j \Psi_j \, dV \tag{3.39}$$

$$\mathbb{M}_{ij}^{au} = \rho \int_V \Psi_i \Phi_i \Phi_j \, dV \tag{3.40}$$

$$\mathbb{M}_{ij}^{aa} = \rho \int_V \Psi_i \Phi_i \Phi_j \Psi_j \, dV \tag{3.41}$$

The lumped submatrices $\mathbb{M}_{ij}^{ua}$ and $\mathbb{M}_{ij}^{au}$ are zero, whereas the submatrices $\mathbb{M}_{ij}^{uu}$ and $\mathbb{M}_{ij}^{aa}$ can be diagonalized using row summation (cp. equation (2.34))

$$\bar{\mathbb{M}}_{ii}^{uu} = \sum_j \mathbb{M}_{ij}^{uu} \tag{3.42}$$

$$\bar{\mathbb{M}}_{ii}^{aa} = \sum_j \mathbb{M}_{ij}^{aa} \tag{3.43}$$

or weighted diagonal technique

$$\bar{\mathbb{M}}_{ii}^{uu} = m \frac{\mathbb{M}_{ii}^{uu}}{\sum_j \mathbb{M}_{ij}^{uu}} \tag{3.44}$$

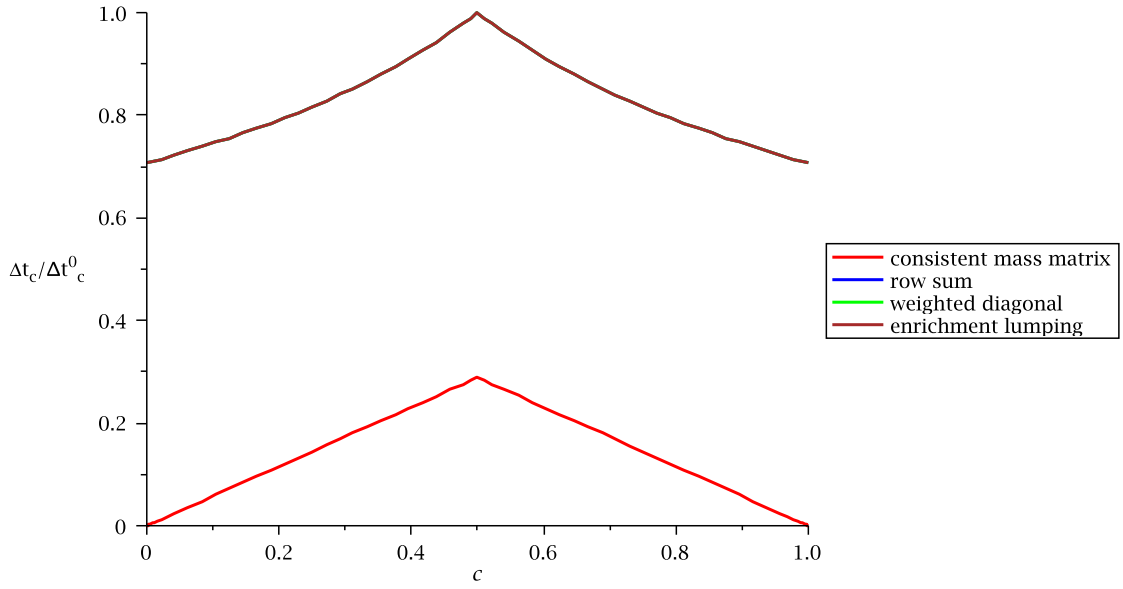
$$\bar{\mathbb{M}}_{ii}^{aa} = m \frac{\mathbb{M}_{ii}^{aa}}{\sum_j \mathbb{M}_{ij}^{aa}} \tag{3.45}$$

[MRCB06] propose a third mass lumping technique specific to XFEM.

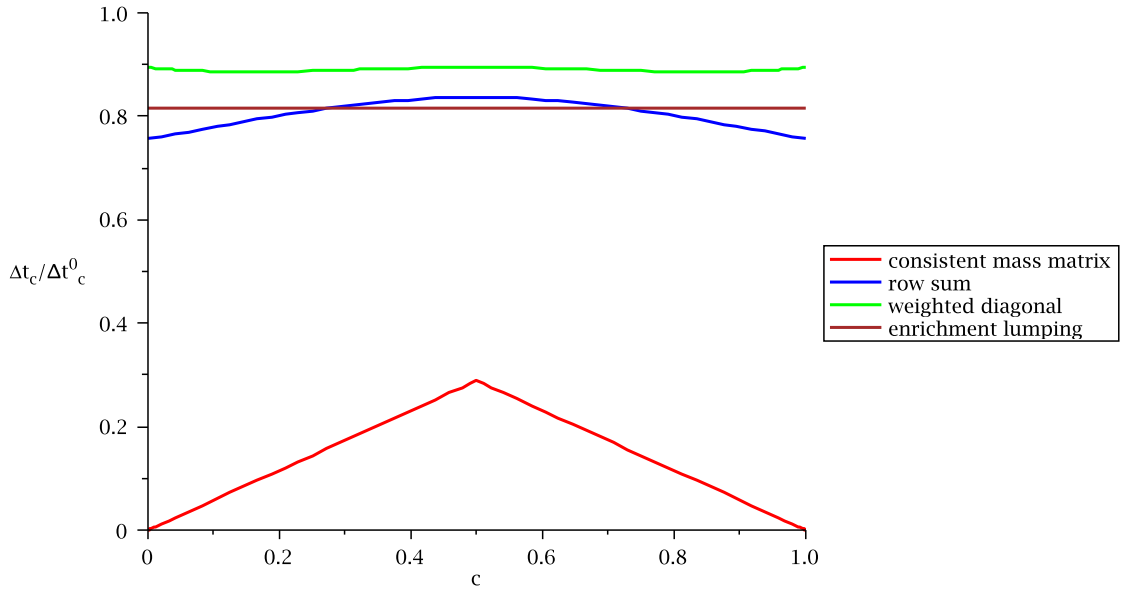
$$\bar{\mathbb{M}}_{ii}^{uu} = \frac{m}{n} \tag{3.46}$$

$$\bar{\mathbb{M}}_{ii}^{aa} = \frac{\rho}{n} \int_V \psi_i^2 \, dV = \left(\frac{V_a}{V} \Psi_{ai}^2 + \frac{V_b}{V} \Psi_{bi}^2 \right) \bar{\mathbb{M}}_{ii}^{uu} \tag{3.47}$$

where m is the element total mass and n is the number of element nodes. They show that for the enrichment function in form of a Heaviside function (0 or 1) or



(a) generalized Heaviside function



(b) shifted enrichment function

Figure 3.10: The stability analysis of a dissected 1D element simulated by XFEM. The horizontal axis c denotes the position of the cut relative to the total element length.

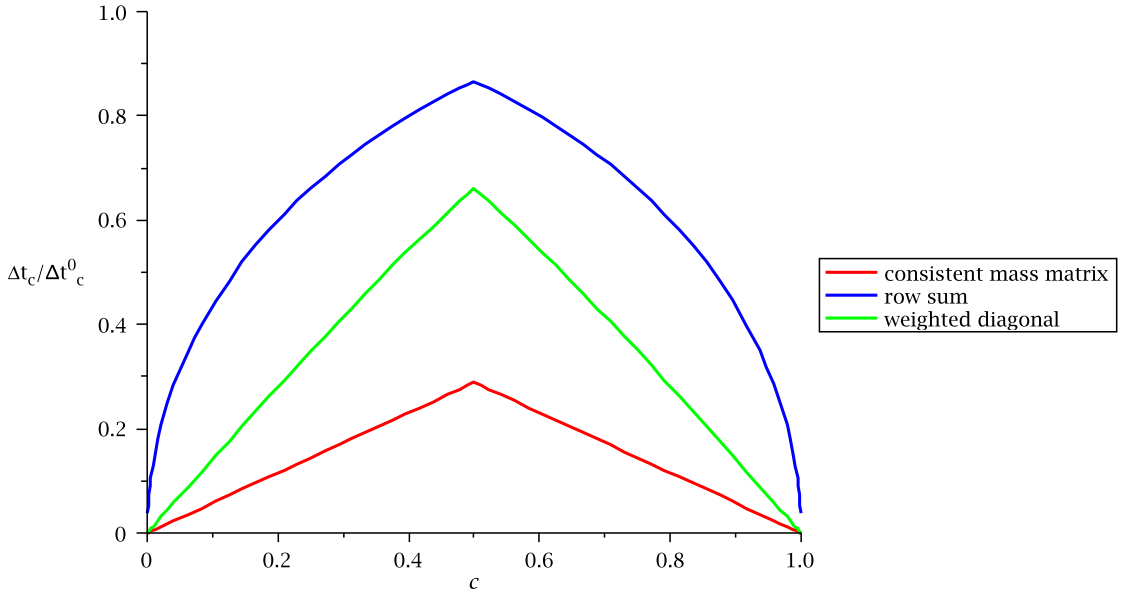


Figure 3.11: The stability analysis of a dissected 1D element simulated using DOF substitution. The horizontal axis c denotes the position of the cut relative to the total element length.

a generalized Heaviside function (-1 or +1) their mass lumping is superior to the previous two as the critical time step never becomes zero, regardless of the position of the cut.

I analyzed the stability criterion for a one dimensional element enriched by the generalized Heaviside function (cp. equation (3.2)), the shifted enrichment function (cp. equation(3.3)) and the DOF substitution (cp. equation (3.5)) with the cut position as a parameter. Figures 3.10 and 3.11 show the analysis results. The vertical axis is the ratio of the critical time step of an enriched element Δt_c and a standard element with row summation mass lumping Δt^0_c. The values of this ratio are expected to be in the range [0,1], where 0 means instability (i.e., $\omega = \infty$). On the horizontal axis is the position of the cut relative to the total length. I compare the enriched consistent matrix and the three mass lumping techniques.

The consistent matrix will lead to instability when the cut is close to an element node (element sliver). In case of the generalized Heaviside function (cp. Fig. 3.10(a)) all three mass lumping techniques lead to the same diagonal mass matrix. The critical time step of the cut element is between 0.7 and 1.0 of the critical time step of the nonenriched element with the lowest values at both ends of the element. For the shifted enrichment function (cp. Fig. 3.10(b)) the different mass lumping techniques lead to different matrices. The critical time steps reached are higher than 0.75 of

the critical time step of the nonenriched element. In this case the weighted diagonal technique gains slightly better results than the other two lumping techniques.

The critical timestep of an element enriched by the generalized Heaviside function or the shifted enrichment function not only never drops down to zero, but it is of the same order of magnitude as the critical timestep of a standard element regardless of the cut location. The situation is different when the DOF substitution is used (cp. Fig. 3.11). The row summation and weighted diagonal mass lumping techniques both lead to instability for a cut near the element nodes. The enrichment lumping cannot be applied here.

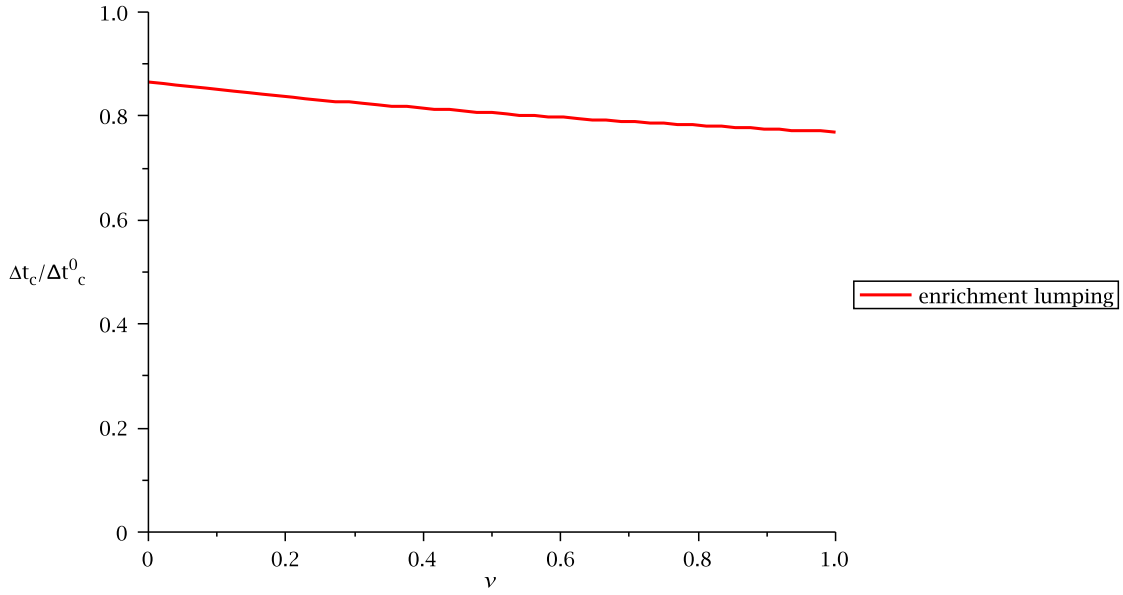


Figure 3.12: The stability analysis of a dissected tetrahedron simulated by XFEM using shifted enrichment function and a XFEM specific mass lumping. The horizontal axis v denotes the volume below the cut relative to the total element volume.

For tetrahedra in 3D space the general analysis becomes far more complex as the size of the involved matrices grows from $[4 \times 4]$ in 1D case to $[24 \times 24]$ in 3D case. Moreover, the cut position has to be parametrized using three independent parameters. For the consistent matrix and the first two mass lumping approaches a volume integral of the shape functions above and below the cut has to be evaluated first. Then the mass lumping is applied and the resulting matrix is inverted. This leads to a nontrivial dependency of the eigenvalues on the cut parameters making a generic numerical analysis too difficult for a state of the art mathematical software. In case of the enrichment lumping (cp. equation (3.47)) the values of the mass matrix only depend on the volume ratios above and below the cut and thus only one parameter defining the cut position is sufficient. Fig. 3.12 shows the result of the eigenvalue analysis for a tetrahedron enriched by the shifted enrichment function.

The vertical axis is the ratio of the critical time step of an enriched element and a standard element with row summation mass lumping. On the horizontal axis v is the volume below the cut relative to the total volume of the element. As in 1D case, also in 3D case the critical time step of an enriched element is of the same order of magnitude as the critical time step of a nonenriched element regardless of the location of the cut. Consequently, situations as displayed in Fig. 3.9 can be simulated without problems.

3.3 Complexity Analysis

The linear, corotational and geometrically nonlinear methods described above are suitable for a real time simulation of deformable objects in virtual environments. In this section, the impact of cutting on the simulation performance is analyzed. The deformation forces and their derivatives can be evaluated for each element separately. I analyze the computational overhead of an enriched element over a standard element. For the linear constitutive model the stiffness matrix is precomputed at the beginning of the simulation, and the enriched stiffness matrix is computed once the position of the discontinuity is known. The nodal deformation force corresponding to a standard DOF of an enriched element ($0 \leq i \leq n$) is computed as

$$\mathbf{f}_i = \sum_{j=1}^{n} \mathbb{K}_{ij}^{uu} \mathbf{u}_j + \sum_{j=1}^{n} \mathbb{K}_{ij}^{ua} \mathbf{a}_j \tag{3.48}$$

The nodal deformation force corresponding to an added DOF is computed as

$$\mathbf{f}_i^a = \sum_{j=1}^{n} \mathbb{K}_{ij}^{au} \mathbf{u}_j + \sum_{j=1}^{n} \mathbb{K}_{ij}^{aa} \mathbf{a}_j \tag{3.49}$$

It can easily be seen that the force evaluation of an enriched element requires approximately four times more floating point operations than a standard element. The forces Jacobian components

$$\mathbb{J}_{ij}^X = \frac{\partial \mathbf{f}_i^X}{\partial \mathbf{u}_j^X} \tag{3.50}$$

are identical to the stiffness matrix components and thus remain constant during the simulation.

For the enriched corotational elements the two rotation matrices $\mathbb{R}_a$ and $\mathbb{R}_b$ for the parts above and below the cut have to be determined. The nodal deformation forces are computed according to equations (3.24) and (3.27). Unlike the linear case, the forces Jacobian is not constant. However, during the computation of the

deformation forces some terms can be reused and can also be used in the Jacobian computation. The force evaluation of an enriched element requires approximately 2.6 times as many floating point operations as the nonenriched corotational element. The forces Jacobian is computed as (cp. equation (3.23))

$$\mathbb{J}_{ij}^{X} = \underbrace{\mathbb{R}_a \int_{V_a} \mathbb{B}_i^{XT} \mathbb{C} \mathbb{B}_j^{X} \, dV \, \mathbb{R}_a^T}_{above} + \underbrace{\mathbb{R}_b \int_{V_b} \mathbb{B}_i^{XT} \mathbb{C} \mathbb{B}_j^{X} \, dV \, \mathbb{R}_b^T}_{below} \tag{3.51}$$

resulting in

$$\mathbb{J}^{X} = \begin{bmatrix} \mathbb{J}^{uu} & \mathbb{J}^{ua} \\ \mathbb{J}^{au} & \mathbb{J}^{aa} \end{bmatrix} \tag{3.52}$$

$$\mathbb{J}_{ij}^{uu} = \frac{V_a}{V} \mathbb{R}_a \, \mathbb{K}_{ij} \, \mathbb{R}_a^T + \frac{V_b}{V} \mathbb{R}_b \, \mathbb{K}_{ij} \, \mathbb{R}_b^T \tag{3.53}$$

$$\mathbb{J}_{ij}^{ua} = \Psi_{aj} \frac{V_a}{V} \mathbb{R}_a \, \mathbb{K}_{ij} \, \mathbb{R}_a^T + \Psi_{bj} \frac{V_b}{V} \mathbb{R}_b \, \mathbb{K}_{ij} \, \mathbb{R}_b^T \tag{3.54}$$

$$\mathbb{J}_{ij}^{au} = \Psi_{ai} \frac{V_a}{V} \mathbb{R}_a \, \mathbb{K}_{ij} \, \mathbb{R}_a^T + \Psi_{bi} \frac{V_b}{V} \mathbb{R}_b \, \mathbb{K}_{ij} \, \mathbb{R}_b^T \tag{3.55}$$

$$\mathbb{J}_{ij}^{aa} = \Psi_{ai} \Psi_{aj} \frac{V_a}{V} \mathbb{R}_a \, \mathbb{K}_{ij} \, \mathbb{R}_a^T + \Psi_{bi} \Psi_{bj} \frac{V_b}{V} \mathbb{R}_b \, \mathbb{K}_{ij} \, \mathbb{R}_b^T \tag{3.56}$$

The nodal deformation forces of a geometrically nonlinear enriched element are computed using equations (3.34) and (3.35). The deformation gradient, strain and stress are determined only once per simulation time step for each dissected part and reused in the computation of all nodal forces. Thus, the force evaluation of an enriched element requires approximately 2.2 times more floating point operations than a nonenriched nonlinear element. The rows of the forces Jacobian corresponding to the standard DOF ($0 \leq i \leq n$) are computed as

$$\mathbb{J}_{ij}^{uX} = \beta_i \left(V_a \frac{\partial \left(\mathbb{S}_a \mathbb{F}_a^T \right)}{\partial \mathbf{u}_j^X} + V_b \frac{\partial \left(\mathbb{S}_b \mathbb{F}_b^T \right)}{\partial \mathbf{u}_j^X} \right) \tag{3.57}$$

whereas the rows corresponding to the added DOF are computed as

$$\mathbb{J}_{ij}^{aX} = \beta_i \left(\Psi_{ai} V_a \frac{\partial \left(\mathbb{S}_a \mathbb{F}_a^T \right)}{\partial \mathbf{u}_j^X} + \Psi_{bi} V_b \frac{\partial \left(\mathbb{S}_b \mathbb{F}_b^T \right)}{\partial \mathbf{u}_j^X} \right) \tag{3.58}$$

Although the partial derivatives $\frac{\partial \left(\mathbb{S}_a \mathbb{F}_a^T \right)}{\partial \mathbf{u}_j^X}$ and $\frac{\partial \left(\mathbb{S}_b \mathbb{F}_b^T \right)}{\partial \mathbf{u}_j^X}$ are only computed once and reused for each i, their dimension is twice the dimension of $\frac{\partial \left(\mathbb{S} \mathbb{F}^T \right)}{\partial \mathbf{u}_j}$ of a standard element. Thus, compared to the forces Jacobian of a standard nonlinear element,

the update of the enriched element forces Jacobian requires approximately four times more floating point operations.

Remeshing based methods (e.g., [MK00]) replace a cut tetrahedron with 5.2 new elements in average. These methods suffer from serious stability problems caused by small tetrahedra. Therefore, a very small simulation time step has to be chosen when the elements are cut. As can be seen above, the computational costs of an enriched element in XFEM are 2.2 to 4 times the costs of the corresponding standard element depending on the applied constitutive model and integration scheme (explicit schemes do not require the computation of the forces Jacobian). The simulation time step is not influenced by cutting in any way. In summary, XFEM can efficiently model discontinuities within an FEM mesh and is suitable for a real time simulation of cutting as used, e.g., in virtual surgery.

Chapter 4

The Simulation Framework

The methods presented in this thesis are part of a framework for interactive physically based modeling of deformable objects in virtual environments. The PBM framework is based on the VR toolkit ViSTA (Virtual Reality for Scientific and Technical Applications) developed in the VR-Group of the RWTH Aachen University and should enable the integration of PBM into a broad variety of VR applications. Therefore, the simulation API was designed to be general and independent of specific simulation methods. It provides state of the art algorithms for elastic deformation, numerical solvers, collision detection and supporting utilities.

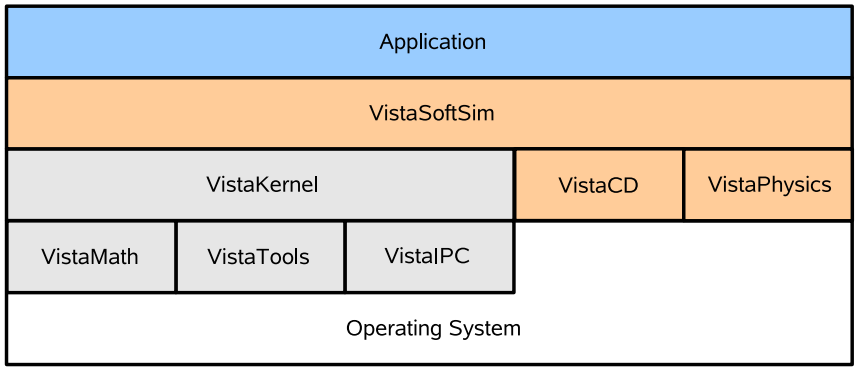


Figure 4.1: The layered structure of a VR application with physically based simulation. The basic ViSTA modules (gray) provide general VR infrastructure whereas the PBM modules (orange) provide methods of collision detection and physical simulation.

The structure of a VR application with physically based simulation is shown in Fig. 4.1. `ViSTAKernel` manages the objects in a scene using a scene graph API.

Moreover, it provides access to display configurations ranging from desktop to cluster based visualization in CAVE-like environments. ViSTA supports various I/O devices as motion trackers, navigation devices and the Phantom haptic device. Modules `VistaMath`, `VistaTools`, `VistaIPC` provide basic functionality that can be accessed by `VistaKernel` as well as by the application. The above mentioned modules provide a general infrastructure for the creation of virtual environments. Special functionality can be provided by additional modules. The PBM modules will be explained in the following sections.

4.1 The Application Backbone

The `VistaSoftSim` module creates a bridge between the application and the `VistaCD` and `VistaPhysics` modules. The purpose of `VistaSoftSim` is the simplification of the use of PBM methods by providing utilities and patterns needed by the application.

An interactive physically based simulation involves the execution and synchronization of the following tasks: *visualization*, *deformation*, *collision detection* and *force feedback*. Each of these tasks requires different data structures and different update rates. The *visualization* renders the polygonal approximation of the surface of the simulated object within a graphical scene. The graphical representation of the interaction tools is visualized as well. A sufficient update rate for a smooth animation is about 20 Hz. A higher frame rate is desirable though when head tracking is used. The *deformation* process uses a mesh of finite elements approximating the volumetric object. Depending on the material parameters and the numerical method used, it has to be run up to several thousand times per second. The deformation process is the most computationally expensive task in the system. The *collision detection* is used to approximate the tested surface and to quickly identify parts of the surface that the tool collides with. The interaction tools can be approximated using one or more line segments. The collision detection should be processed about two hundred times per second. The *force feedback* process is usually run on a dedicated computer with a force feedback device attached to it. Here, a simplified local model of the simulated object is created. The force is proportional to the penetration depth and has to be updated about thousand times per second in order to provide smooth feedback without vibrations. Fig. 4.2 displays a structure of an interactive soft tissue simulator.

The different object representations have to be kept consistent as the object undergoes deformation and topological changes. As the user perceives the system visually and haptically, it is crucial to provide the visual and haptical output at the specified rates. The collision data are provided to the haptics process immediately at the rate

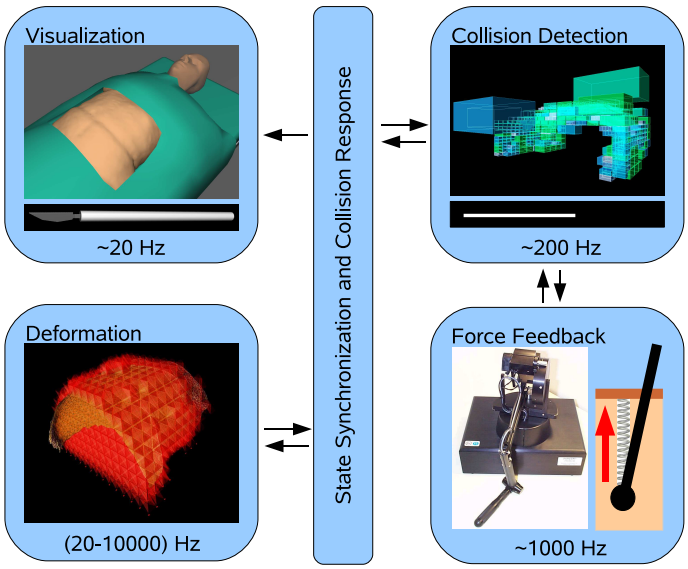


Figure 4.2: The structure of an interactive soft tissue simulator. Each task uses a dedicated representation of the simulated object and surgical tool.

of the collision detection. The data between the visualization, deformation and collision detection are synchronized at a lower rate given by the *world time step*, which is a multiple of the simulation time step used by the deformation process. The world time step must not be smaller than 50 ms, which corresponds to an update rate of 20 Hz. Any user interaction detected between two synchronization steps is queued and processed simultaneously before the resulting changes are visualized.

Fig. 4.3 displays the synchronization mechanism. The crucial part is to realize the data exchange fluently and without too much friction between the submodules. E.g., the frame rate for the visualization must remain constant in order to realize a smooth projection update for the user's head position. For that purpose I propose a synchronization protocol that is described as follows. The visualization and collision detection threads run in a loop that can be paused by a synchronization event sent by another thread. A time out (t.o.) is used to control the update rate of these threads. The synchronization is controlled by the deformation thread, which sends a synchronization event to the visualization thread every time a world time step has been simulated. Once the deformation and visualization thread pass the B1 barrier, a synchronization event is sent to the collision detection thread. After the B2 barrier has been passed, all threads are paused and the synchronization can be executed. While the visualization thread controls the data exchange, the collision detection and the deformation threads are waiting to be released at barriers B3 and B4 respectively.

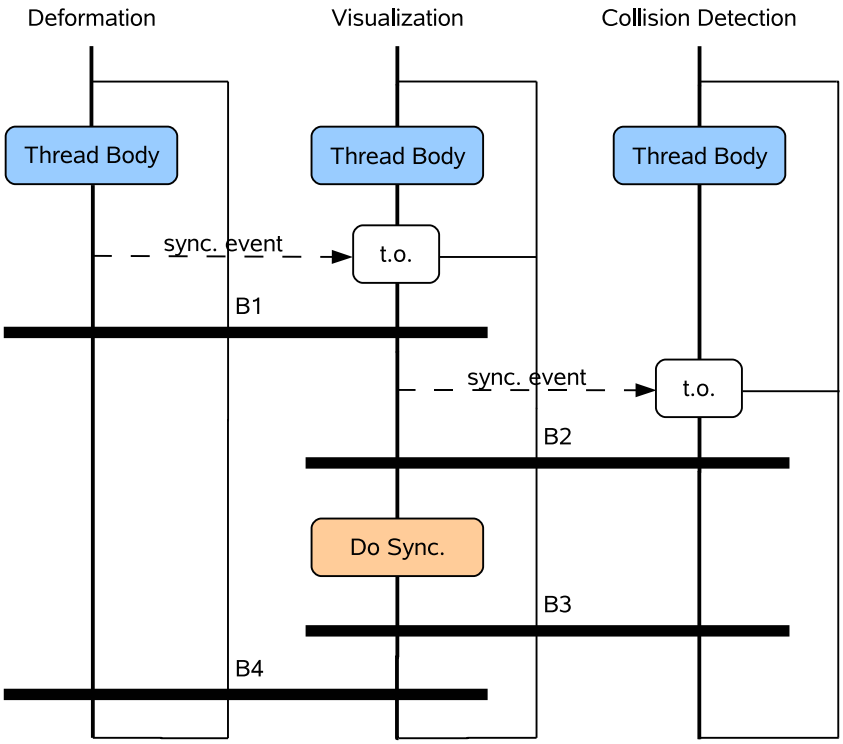


Figure 4.3: The synchronization mechanism.

4.1.1 Mesh Generation

In addition to the synchronization of the crucial simulation tasks, the `VistaSoftSim` module provides algorithms for the mesh generation and (de-)serialization. A FE mesh can be generated for an arbitrary geometry using hexahedral or tetrahedral volumetric elements or springs. The volumetric mesh generation in `VistaSoftSim` is similar to [THMG04] and [SWT06]. The resolution of the FE mesh is independent of the resolution of the geometry surface. Practical experience shows, that the visualization mesh has to be of a high resolution in order to achieve an appealing result, whereas an FE mesh of much lower resolution provides a sufficient amount of information to animate a detailed surface mesh [MG04, MBF04, JKWP04]. This leads to two coupled representations of the simulated object. The resolution of the FE mesh can be chosen depending on the application requirements and the computational power available for the simulation. In the FEM, the deformation field is given by an interpolation of the nodal displacements within the elements using shape functions. For a linear tetrahedron, this corresponds to the linear interpolation using barycentric coordinates. Each vertex of the detailed geometry surface is assigned to a tetrahedron and the barycentric coordinates of the vertex are computed in the non-deformed state. The barycentric coordinates remain constant during the simulation and as the FEM mesh deforms, the positions of the geometry vertices

are interpolated accordingly (cp. Fig. 4.4). The resulting mesh and the mapping between its elements and the original geometry can be used in the application or saved in a file for later use.

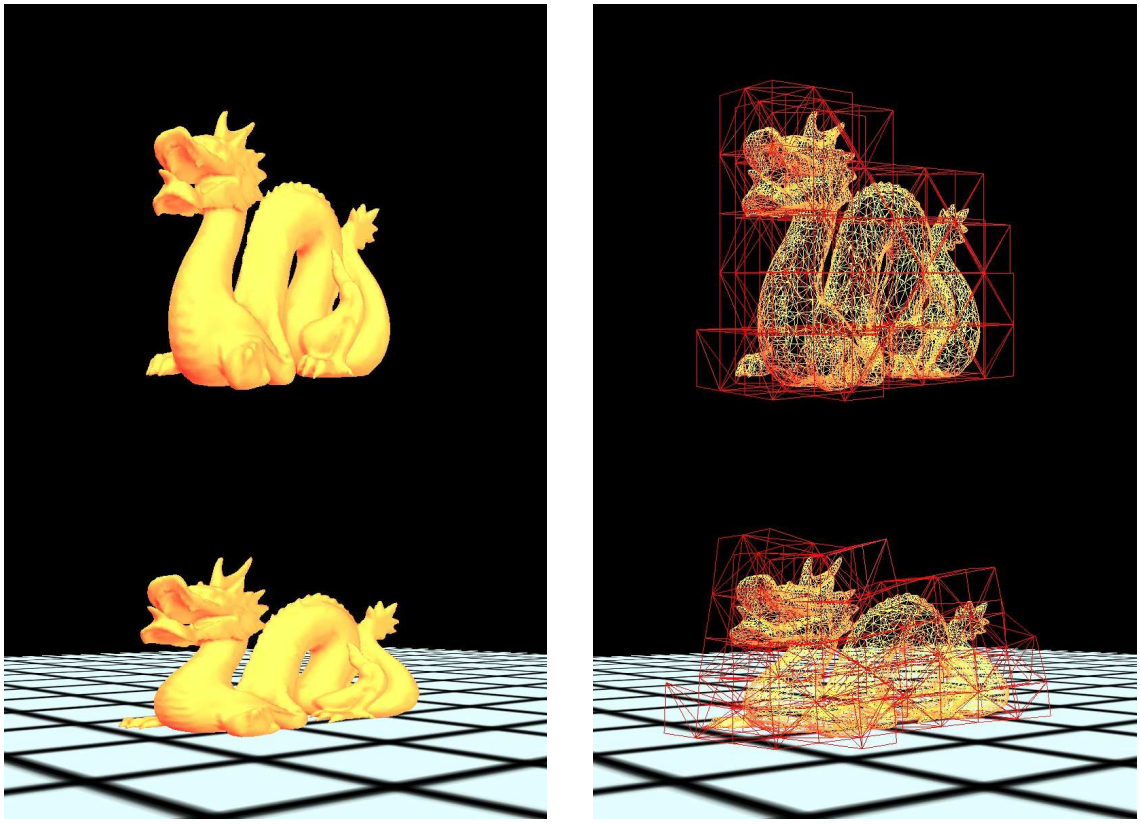


Figure 4.4: The deformation mesh (red wireframe) can use a significantly lower resolution than the visualized geometry (yellow) while achieving acceptable results.

In case of medical data, both the FE mesh and the geometry surface are generated from patient data in a time demanding preprocessing involving the following steps [BOW+00]:

- Obtain medical images and texture maps. Medical images include MRI, CT or ultrasound scans.

- Extract tissue structure contours (segmentation). Although there are some software packages for semiautomatic segmentation, this step still requires a high amount of manual manipulation.

- Generate a 3D mesh from the tissue structure contours. The tissue contours are converted into implicit solids. A closed boundary surface is generated for

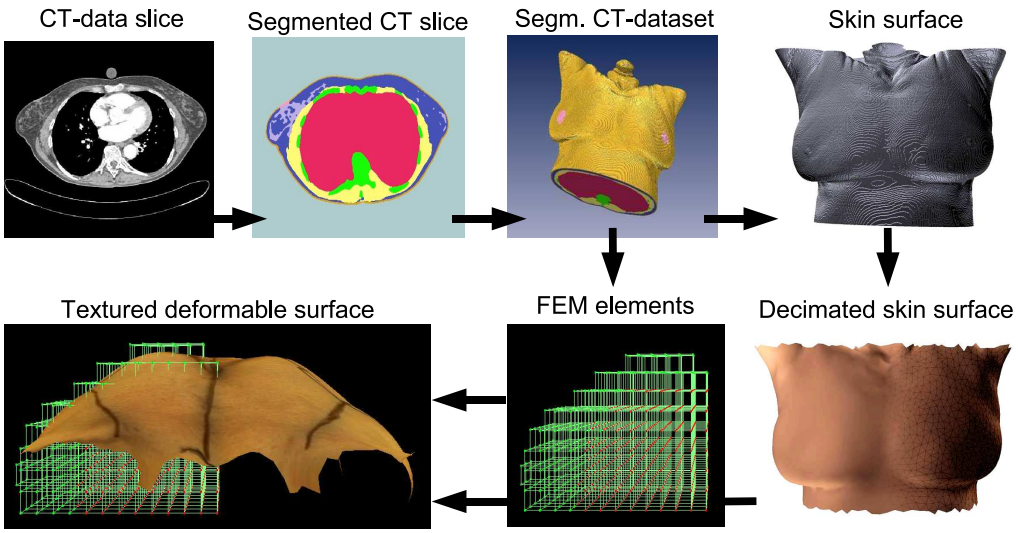


Figure 4.5: Data preprocessing.

each tissue structure. Each surface is filled with equally distributed nodes. Finally, the Delaunay triangulation is used to create a tetrahedral mesh.

[MT03] present an approach to volumetric mesh generation suitable for real-time physically based medical simulations. In both [MT03] and [BOW+00] approaches the geometry surface used for visualization is identical to the boundary of the tetrahedral mesh. All vertices of the geometry are placed at corresponding FEM mesh nodes.

In my approach the resolutions are independent (cp. Fig. 4.5) allowing for smooth surface generation and efficient simulation at the same time. Moreover, a hierarchical Octree-based FE mesh can be generated as described in [JKWP04].

- Altering mesh based on simulation objectives. In this step the FE mesh can be tailored to the application needs, for example a wound or a cancerous tissue can be created.

- Assign material properties, fix boundary nodes in space and assign texture maps.

4.1.2 Multiresolution Deformation

A realistic tissue deformation at the area of intervention is of high importance for a surgery training system. At the same time, a real time response to user actions

in an interactive virtual environment is crucial. Organizing the finite elements in a hierarchical structure enables an adaptive refinement of the mesh at the area of interest. The simulation starts at a default resolution level. Once the user interacts with the object, the mesh is gradually refined using a fine resolution at the area of interaction and coarse resolutions at distant areas. Further refinement criteria such as the velocity of the FEM nodes or a local deformation error can be specified.

I proposed an octree based approach in [JKWP04]. When an FEM element is refined, it is replaced by 8 elements with a half edge length of the original element and new FEM nodes are added as well. The octree contains the topological structure of the FEM elements. Figure 4.6 shows a deformation of a bar using three element levels. There are three types of nodes. The red colored nodes are fixed, the green colored nodes can be updated by a standard FEM computation at a certain level. The yellow colored nodes were added to the mesh during a refinement procedure, but can not be updated using the fine level stiffness matrix values, because some of their adjacent elements and thus neighboring nodes on the fine level are still missing. These nodes always lie on an edge between two coarse level nodes or in the middle of a coarse level cube face. Their position is interpolated using the shape functions of the coarser mesh level. In order to preserve the conformity of the mesh [ZTZ05], the level of neighboring FEM elements has to be the same or different by one.

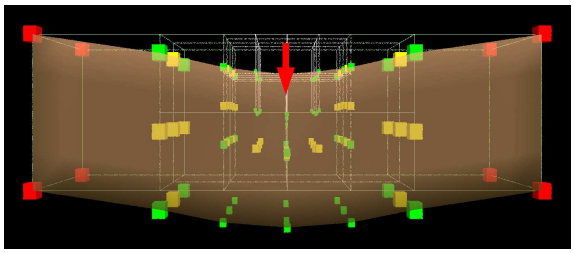


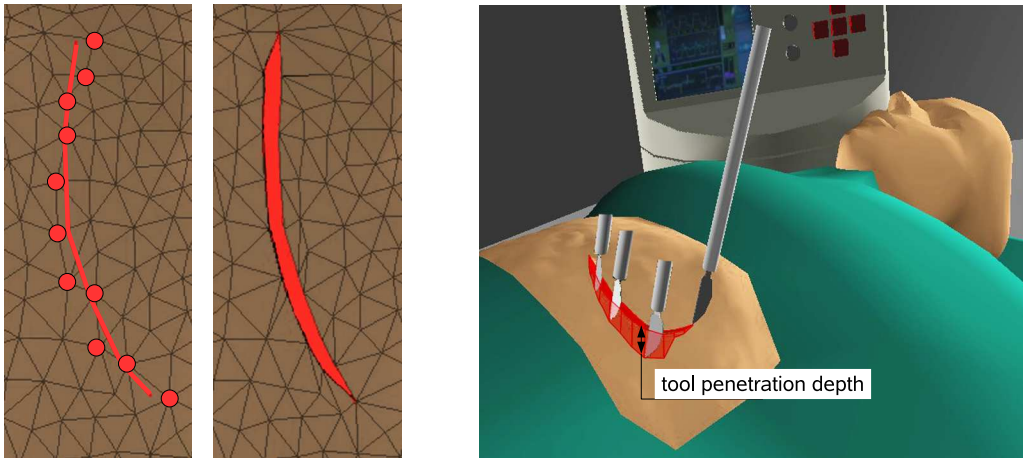
Figure 4.6: The deformation of a bar using three different FEM element sizes. A force acts at the center of the top side of the bar (red arrow), the left and the right sides are fixed.

Related approaches were presented by [GKS02] and [NFP06]. [GKS02] define the hierarchy in terms of hierarchical shape functions. Fine resolution levels are added by activating shape functions belonging to the corresponding level. For cube elements the results are the same as using the octree data structure. However, the shape functions refinement can be also applied to other element types. This concept offers a strong potential especially in the combination with the XFEM used for cutting (see chapter 3).

4.1.3 Geometry Update during Cutting

The continuous deformation field is given by equation (2.20). For a linear tetrahedron, this corresponds to the linear interpolation using barycentric coordinates. Each vertex of the detailed geometry surface is assigned to a tetrahedron and the barycentric coordinates of the vertex are computed in the non deformed state. The barycentric coordinates remain constant during the simulation and as the FEM mesh deforms, the positions of the geometry vertices are interpolated accordingly.

When a cut is created, the deformation field contains a discontinuity. The discontinuity has to be modeled in the surface mesh as well. This can be achieved either by subdividing the triangles intersected by the cut or by snapping the existing vertices to the cut. The first method is more suitable for simulating fractures with small material slivers (e.g., [MBF04]). For surgery simulation we prefer the latter method as it does not generate unnecessary faces along the cut. Both methods lead to triangles, that are either completely above or completely below the discontinuity. The vertices on the cut can be doubled and assigned to the respective side in order to create the opening.



(a) (b) (c)

Figure 4.7: Creating the visual representation of an incision. (a) the triangular mesh of the object surface (blue) with the desired cutting path (red line) and the vertices to be snapped to the cut (red points), (b) the situation with the cut completed, (c) the tool penetration depth is used to model the depth of the wound.

The marked vertices (cp. Fig. 4.7 a) are projected orthogonally on the cut and their new barycentric coordinates are computed. The first and last vertex are snapped exactly to the start and end positions of the cut respectively, instead of being projected orthogonally. At this stage, the geometry contains a sequence of edges between the

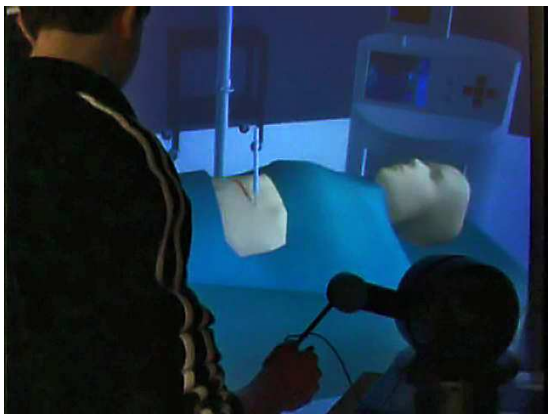
first and last marked vertices, that have to be doubled in order to create the desired topological opening in the geometry. The split interior vertices are still on the same positions as the originals, the created hole is opened by the physical simulation (cp. Fig. 4.7 b).

If surfaces (e.g. cloth) or thin shells were simulated, the cutting procedure would finish at this step. In surgery simulation, however, volumetric objects are simulated. Therefore, the hole in the surface has to be closed properly by modeling a wound. The collision detection routine provides not only the collision points, but also the tool penetration depth at these points. In order to generate forces leading to the deformation or cut of the simulated object, the tool tip has to penetrate the object. The interaction forces are proportional to the penetration depth. The penetration is not displayed visually, the visual representation of the tool remains on the object surface giving the user the feeling of a resistant surface. As the cut path on the surface and its depth below the surface is known, the wound can be modeled by moving a copy of the surface vertices involved in the cut according to the penetration depth. New faces are created between the vertices on the surface and below the surface (cp. Fig. 4.7 c).

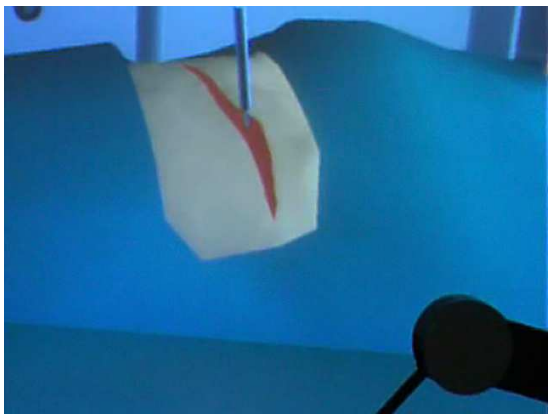
The described technique is suitable for semi-progressive cutting, in the sense that the cut is processed during the interaction as soon as the cutting tool leaves an FEM element. Fully progressive cutting, where the cut is created immediately up to the current tool position requires a significant computational overhead [MK00]. Semi-progressive cutting is a generally accepted alternative. This method has been integrated into a surgical simulator (cp. Fig. 4.8) [JK07]. Incisions can be created interactively with force feedback, which helps the user to control the incision depth. Once an incision has been created, it can be manipulated (e.g., opened) in order to perform a specific surgical procedure (cp. Fig. 4.8 b).

4.2 Collision Detection

Collision detection (CD) is used to support interaction of virtual objects with each other and user manipulation tasks. The often used methods for speeding up the collision detection are all kinds of Bounding Volumes (BV), e.g., oriented bounding boxes (OBB) [GLM96], axis aligned bounding boxes (AABB) [vdB97], bounding spheres (BS) [Hub96], binary space partitioning (BSP) trees [BV91] or Octrees [FP02]. All these methods were originally developed for collision detection of rigid bodies. Deformable objects complicate the problem, as the objects and thus the collision data structures have to be updated frequently during the simulation as opposed to initializing them once in a preprocessing step. Furthermore, rigid body simulations often neglect self-collisions and multiple contacts, which have to be considered in



(a) Interactive cutting is enhanced by force feedback.



(b) The incision can be manipulated in order to perform a specific surgical procedure.

Figure 4.8: Creating an incision in a virtual surgery simulator.

order to simulate the interaction with deformable objects realistically. A realistic collision response requires appropriate information. It is not sufficient to just detect the interference of the objects. More precise information such as penetration depth is desired.

[TKH+04] summarize recent research in the area of deformable CD. Various approaches based on BV hierarchies, distance fields, spatial partitioning, image space techniques and stochastic methods are discussed. [THM+03] propose an approach to collision and self collision detection of dynamically deforming objects that consist of tetrahedrons. The proposed algorithm employs a hash function for compressing a potentially infinite regular spatial grid. Although the hash function does not always provide a unique mapping of grid cells, it can be generated very efficiently and does not require complex data structures, such as trees. Although the algorithm works with tetrahedral meshes, it can be easily adapted to other object primitives, such as triangles.

As the CD research offers many state of the art algorithms, `VistaCD` module was designed to make them available in a common interface. The following collision detection methods are provided: Octree, AABB hierarchy, BS hierarchy, spatial hashing. During a contact between the simulated object and a tool, the CD determines discrete points of contact and the tool penetration depth at these points. This data is used by the haptics in order to give the user the feeling of a resistant surface. Moreover, the collision data is used to generate interaction forces leading to a local deformation of the simulated object.

4.3 Deformation Core

From the point of view of this thesis, `VistaPhysics` is the most relevant module. The `VistaPhysics` module is independent of other modules and does not provide any visualization or interaction methods. It contains the implementation of state of the art methods for the simulation of elastic deformation, optimized vector and matrix data structures and the required numerical methods. Fig. 4.9, 4.10, 4.11 display the class diagrams of the most relevant components of the `VistaPhysics` module. The classes in figure 4.9 refer to the deformable object as a whole.

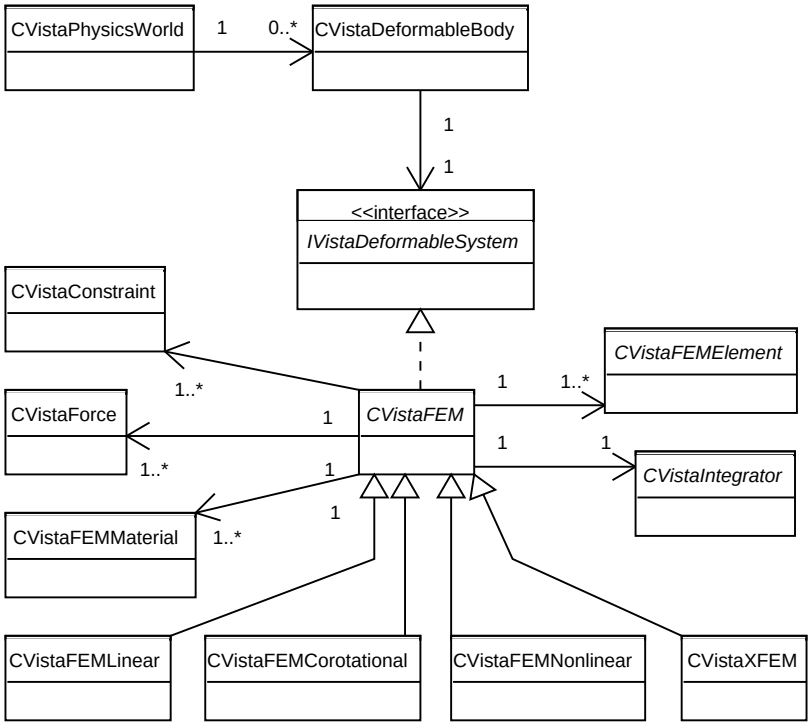


Figure 4.9: `VistaPhysics` class diagram I: the deformable body level.

The `CVistaPhysicsWorld` can contain one or more deformable bodies. Moreover, it defines global world parameters such as gravity or the world time step. From the application point of view, the deformable body is represented by a surface mesh. The application programmer specifies a method that has to be used to compute the body deformation without the need of a deeper knowledge of the chosen method. Typically, the deformation methods use another mesh (of volumetric elements or springs) representing the same object. The `CVistaDeformableBody` defines the mapping between the two representations. The application only knows the surface mesh, whereas the classes implementing the `IVistaDeformableSystem` interface only know their own internal representation. The `IVistaDeformableSystem` de-

fines the generic interface of a deformation method. The `CVistaFEM` is an example of a specific deformation method. It stores the lists of external forces and constraints that have to be applied to the simulated object. Moreover, it maintains the object's material table and holds a reference to a `CVistaIntegrator` for the purpose of dynamic simulation. The FEM represents the deformable object using finite elements, therefore the `CVistaFEM` object stores the mesh of `CVistaFEMElement`s, the global state vector (nodal displacements and velocities), the global forces vector and (optionally) the global stiffness matrix. Various alternatives of the FEM exist. They mainly differ in the way the element internal forces are computed whereas the mesh assembly principle remains the same. The specifications of the abstract `CVistaFEM` class are responsible for creating the appropriate element type.

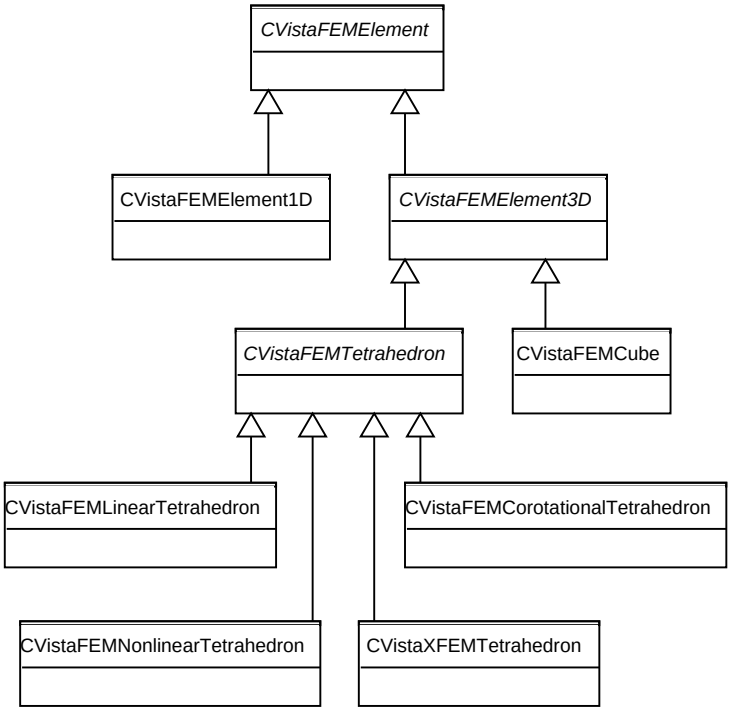


Figure 4.10: `VistaPhysics` class diagram II: the finite elements level.

Fig. 4.10 shows the different finite element types. Each element derived from the `CVistaFEMElement` stores the global IDs of its nodes, material parameters and the element stiffness matrix. The `CVistaFEMElement1D` corresponds to a simple elastic string connecting two mesh nodes. As for volumetric elements, `VistaPhysics` implements hexahedral and tetrahedral elements corresponding to different FEM approaches. Each element must be able to evaluate the deformation forces corresponding to the current element state and their partial derivatives with respect to nodal displacement and velocity. Moreover, each element is able to interpolate the value of displacement anywhere within itself.

The dynamic simulation requires numerical integration. `CVistaIntegrator` is an abstract class providing a generic integrator interface consisting of a single method `Integrate(const State& currentS, State& newS)`. A variety of numerical integration schemes implementing this interface is provided (cp. Fig. 4.11 only displays some of them).

The state of a deformable object is described by a vector of its nodal displacements and velocities. These vectors contain n entries, where n is the number of the object's nodes and each entry is a 3D vector. The vectors of global internal and external forces have the same structure as well. Similarly, the stiffness matrix has the dimensions $n \times n$, where n is the number of element's nodes in case of an element stiffness matrix or the number of the object's nodes in case of the global stiffness matrix. Each entry is a 3×3 matrix. The element stiffness matrix is dense, whereas the global stiffness matrix is a sparse symmetric matrix. `VistaPhysics` contains classes covering the described vector and matrix cases (cp. Fig. 4.11) and the corresponding mathematical operations. The `CVistaLASolver` collects methods of linear algebra, such as the conjugate gradients (CG) solver, a generic matrix inversion and other utilities.

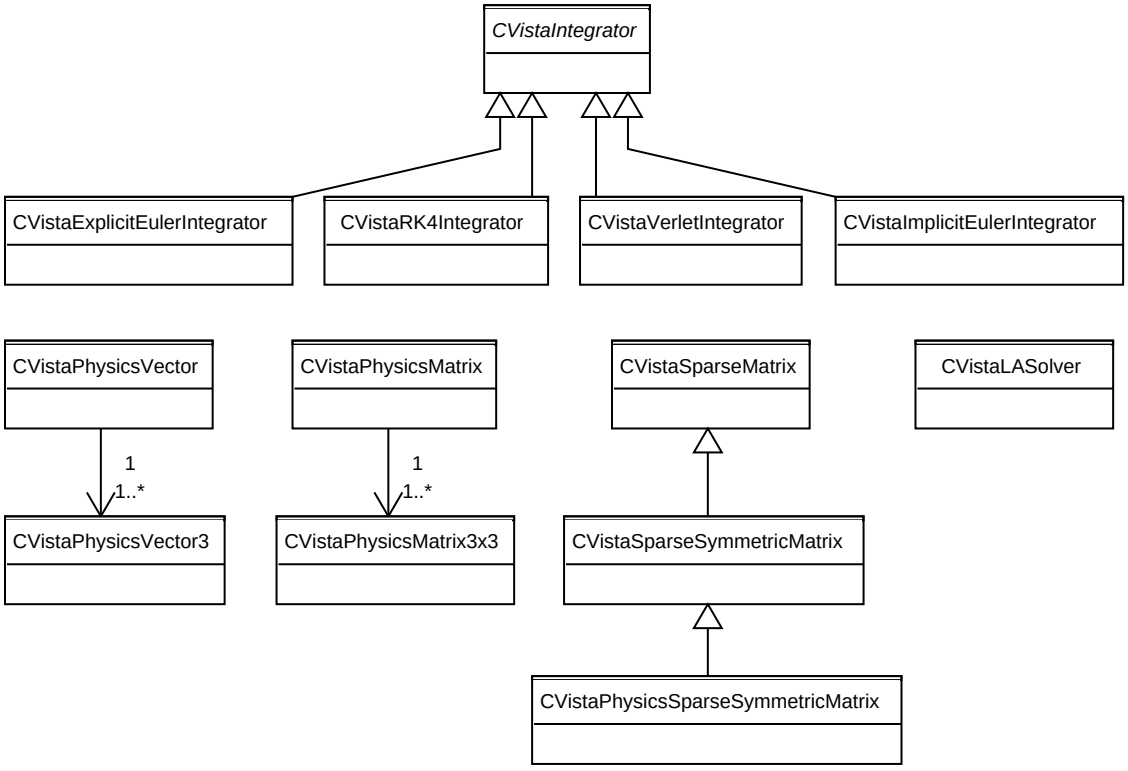


Figure 4.11: `VistaPhysics` class diagram III: math classes.

The `VistaPhysics` module, as well as other ViSTA modules, aims to be platform independent. Therefore, the dependency on external libraries has to be considered carefully. Even though a number of optimized math libraries is available, they are seldomly platform independent and they mostly concentrate on one group of numerical algorithms, e.g., linear algebra with dense matrices, specialized methods for sparse matrices, differential equation solvers, nonlinear solvers providing an exhausting amount of sophisticated methods. `VistaPhysics` requires a mixture of methods from different areas of numerical mathematics, though, it only requires a small fraction of them, most of them being easy to understand and to implement. Moreover, some algorithms as, e.g., the Newton-Raphson or CG must be slightly modified in order to account for physical constraints (see [AB03] for more details). For these reasons, the dependency on external math libraries has been avoided.

All data structures and operations use single precision float, except for the CG algorithm, which uses double precision for the residual in order to achieve both high precision and high performance. The targeted application of double precision for critical operations of single precision algorithms was proposed and exploited by [LLL+06].

4.4 The Simulation Loop

The thread body of the deformation thread (cp. Fig. 4.3) simulates one world time step. It consists of a (repeated) numerical integration of the object state (cp. Fig. 4.12).

```
Simulate(worldTimeStep)
{
  worldTime += worldTimeStep;
  for all objects do
  {
    while(objectTime < worldTime)
    {
      object->Integrator->Integrate(objectTimeStep);
      objectTime += objectTimeStep;
    }
  }
}
```

Figure 4.12: The thread body of the deformation thread.

The different integration schemes are expressed in terms of the following function calls.

`DerivStateExplicit(currState, stateDeriv)`, which assigns the time derivation of `currState` at the current time to `stateDeriv`

`DerivStateImplicit(currState, stateDeriv, timeStep)`, which assigns the time derivation of `currState` at the end of the specified time step to `stateDeriv`

`AccumState(currState, stateDeriv, newState, timeStep)`, which assigns `newState = currState + timeStep * stateDeriv`

The Euler integration schemes consist of one `DerivState` and one `AccumState` call, where the `timeStep` equals to the `objectTimeStep`. Other integrators (e.g., Midpoint, Runge-Kutta) evaluate the derivations multiple times within an `objectTimeStep` and thus multiple calls of `DerivState` and `AccumState` with different states and different time steps are needed. The integrators are independent of the definition of the `State` and the deformation method. The numerical integration can be applied to any class providing the above interface and a subclass called `State`.

```
CVistaFEM::DerivStateExplicit(const State &state,
                              State &stateDeriv)
{
  ApplyForces(state, forces);
  stateDeriv.pos = state.vel;                 // v_t
  stateDeriv.vel = M^-1 * forces;             // a_t
  ApplyConstraints(state, stateDeriv);
}
```

Figure 4.13: Pseudo code for `DerivStateExplicit`

Fig. 4.13 shows the `CVistaFEM` implementation of the `DerivStateExplicit` function. It corresponds to equations (2.36) and (2.32). The evaluated derivations of displacement and velocity $\mathbf{v}_t$ and $\mathbf{a}_t$ are used in equations (2.37) and (2.38), whose implementation is the `AccumState` function. The major amount of the simulation time is spent for the evaluation of forces. The function `ApplyForces` adds the external (e.g., gravity or user interaction) and internal (deformation) forces to the global force vector. The internal forces are assembled from contributions of all finite elements. Thus, the assembly of the global stiffness matrix can be avoided, saving computational time in particular when the stiffness matrix is changing during the simulation (e.g., due to nonlinearity or topological changes).

Implicit integration schemes use the `DerivStateImplicit` function. Fig. 4.14 shows its implementation in `CVistaFEM`. The `DerivStateImplicit` has to evaluate the derivations of displacement and velocity $\mathbf{v}_{t+\Delta t}$ and $\mathbf{a}_{t+\Delta t}$ to be used in

```
CVistaFEM::DerivStateImplicit(const State &state,
                              State &stateDeriv, float timeStep)
{
  newState = state;                              // start point
  iterate until |s| < epsilon                    // Newton-Raphson
  {
    ApplyForces(newState, forces);
    dfdu = GetForceDerivativeU(newState);
    dfdv = GetForceDerivativeV(newState);
    ApplyConstraints(newState,S);
    bu = newState.u - state.u - timeStep*newState.v;
    bu.MultiplyComponents(S, bu);                // constr(bu)
    b = M *(newState.v - state.v) - timeStep*(forces + dfdu*bu);
    A = M - timeStep*(dfdv + timeStep*dfdu);
    ModifiedCG(A, b, s, S);                      // solve As = b
    newState.v -= s;
    newState.u -= bu + timeStep*s;
  }
  stateDeriv = (newState - state)/timeStep;      // v_t+dt, a_t+dt
}
```

Figure 4.14: Pseudo code for `DerivStateImplicit`

equations (2.41) and (2.42). In order to do that, the `DerivStateImplicit` uses the Newton-Raphson method as described in chapter 2.8.2.

The derivative $\mathbb{J}$ of the equation system (2.43, 2.44) is

$$\mathbb{J} = \begin{bmatrix} \frac{\partial \mathbf{b}_u}{\partial \mathbf{u}} & \frac{\partial \mathbf{b}_u}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{b}_v}{\partial \mathbf{u}} & \frac{\partial \mathbf{b}_v}{\partial \mathbf{v}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \mathbb{M} \end{bmatrix} - \Delta t \begin{bmatrix} 0 & 1 \\ \frac{\partial \mathbf{F}}{\partial \mathbf{u}} & \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \end{bmatrix} \tag{4.1}$$

The iteration steps $\mathbf{s}_u$ and $\mathbf{s}_v$ for $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$ respectively can be computed by solving a linear system of equations.

$$\mathbb{J} \begin{bmatrix} \mathbf{s}_u \\ \mathbf{s}_v \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \end{bmatrix} \tag{4.2}$$

The size of the linear system to be solved can be halved by exploiting the dependencies of the equations.

$$\mathbf{s}_u - \Delta t \cdot \mathbf{s}_v = \mathbf{b}_u \tag{4.3}$$

$$\mathbb{M} \cdot \mathbf{s}_v - \Delta t \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{u}} \cdot \mathbf{s}_u - \Delta t \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \cdot \mathbf{s}_v = \mathbf{b}_v \tag{4.4}$$

$\mathbf{s}_u$ can be expressed from the first equation and inserted into the second equation, which then has only one unknown $\mathbf{s}_v$. Thus, the linear system of equations to be solved is

$$\underbrace{\left[\mathbb{M} - \Delta t \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - (\Delta t)^2 \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right]}_{\mathbb{A}} \cdot \mathbf{s}_v = \underbrace{\mathbf{b}_v + \Delta t \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{u}} \cdot \mathbf{b}_u}_{\mathbf{b}} \tag{4.5}$$

The conjugate gradients (CG) method can be used to solve this system of equations in each simulation step (cp. Fig. 4.15). Once the $\mathbf{s}_v$ has been computed, $\mathbf{s}_u$ can be obtained using equation (4.3). The values of $\mathbf{u}_{t+\Delta t}$ and $\mathbf{v}_{t+\Delta t}$ are then updated as

$$\mathbf{u}_{t+\Delta t}^{i+1} = \mathbf{u}_{t+\Delta t}^{i} - \mathbf{s}_u \tag{4.6}$$

$$\mathbf{v}_{t+\Delta t}^{i+1} = \mathbf{v}_{t+\Delta t}^{i} - \mathbf{s}_v \tag{4.7}$$

The linear system of equations to be solved has the size of $3n$, where n is the number of nodal DOF. The CG method takes an advantage of the sparsity of the system. The sparsity pattern corresponds to the connectivity of the FE mesh. As the number of neighboring nodes is approximately constant over the mesh, the CG algorithm has linear complexity $O(n)$.

```
ModifiedCG(const CVistaPhysicsSparseSymmetricMatrix& A,
           const CVistaPhysicsVector& b, CVistaPhysicsVector& x,
           const CVistaPhysicsVector& S)
{
    b.MultiplyComponents(S,r);            // r = constr(b)
    iterate until |r| < epsilon
    {
      delta = r.Dot(r);

      if(first iteration) beta = 0;
      else beta = delta/delta_old;

      delta_old = delta;
      c.Scale(beta);
      c.Add(r);                           // c = r + beta * c
      A.Multiply(c,q);
      q.MultiplyComponents(S,q);          // q = constr(Ac)
      alpha = delta / c.Dot(q);           // alpha = rr/cq
      c.Scale(alpha);
      x.Add(c);                           // x += alpha * c
      q.Scale(-alpha);
      r.Add(q);                           // r -= alpha * q
    }
}
```

Figure 4.15: Pseudo code for `ModifiedCG`

Both the Newton-Raphson and the CG algorithms slightly differ from the well known methods (see, e.g., [PFTV92]) in that they account for constraints. The results of the standard methods were only valid in an unconstrained space. However, when parts of the simulated object are fixed or interacting with an obstacle (e.g., lying on the floor) it has to be considered in the computation as it strongly impacts the results. The constraints are stored in a vector `S` that is used to project the unconstrained solution to the constrained space in each iteration of both methods. This method was introduced by [BW98] and improved by [AB03].

Assuming constant external forces within a time step, the partial derivation of forces with respect to displacement corresponds to tangential stiffness. When the stiffness matrix can be considered constant (within a time step), then $\frac{\partial F}{\partial u} = \mathbb{K}$ and the whole system is linear and only one Newton-Raphson iteration is needed. The $\frac{\partial F}{\partial v}$ corresponds to the damping matrix $\mathbb{D}$, which is then constant as well (cp. equation (2.35)). Moreover, the expressions for `bu` and `b` are simplified as in the first iteration is `newState=state`.

Up to this point, the numerical algorithms were considered to run on a single processor. Chapter 5 provides a performance analysis of the deformation process and presents a parallelization approach for shared memory architectures as used, e.g., in modern PCs with multicore CPUs.

CHAPTER 5

SHARED MEMORY PARALLELIZATION

In the computing industry, performance increase due to higher clock speed is tapering off. Instead, the computational power is increased by replicating processing units on a single chip (multicore architectures), making parallel programing a necessity for all performance demanding applications.

In order to profit from recent developments in the computing industry (chip level parallelism) and to allow for larger datasets to be handled in real time, I analyzed the runtime behavior of the linear FEM and the corotational approach (cp. sec. 2.5, 2.6). I propose a parallelization based on OpenMP and analyze the scalability of these methods. The presented approach uses several processors or cores in one computer that share the same memory. As will be described in the following, the presented solution requires only minimal changes to the source code and the algorithms do not need to be modified at all. Nonetheless, significant improvements on commodity architectures can be realized.

5.1 Multicore Architectures

The parallelized algorithms were evaluated on two commodity dualcore architectures that basically differ in how they share the on-chip L2 cache. In addition, a prototype of a system with Intel's quad-core architecture was used.

a) AMD Opteron 875 dualcore processors (cp. Fig. 5.1), 2.2 GHz, four of which are grouped in one Sun Fire V40z server. Each core has a 1 MB L2 cache, which

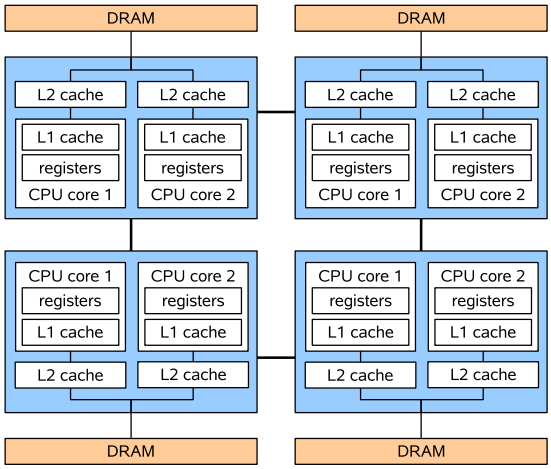


Figure 5.1: The AMD dualcore Opteron system

is not accessible by the other cores. This machine has a ccNUMA architecture where the memory access time depends on the location relative to a processor. On such a system locality is important in order to achieve high performance. The Sun Studio Express C++ compiler under Solaris was employed.

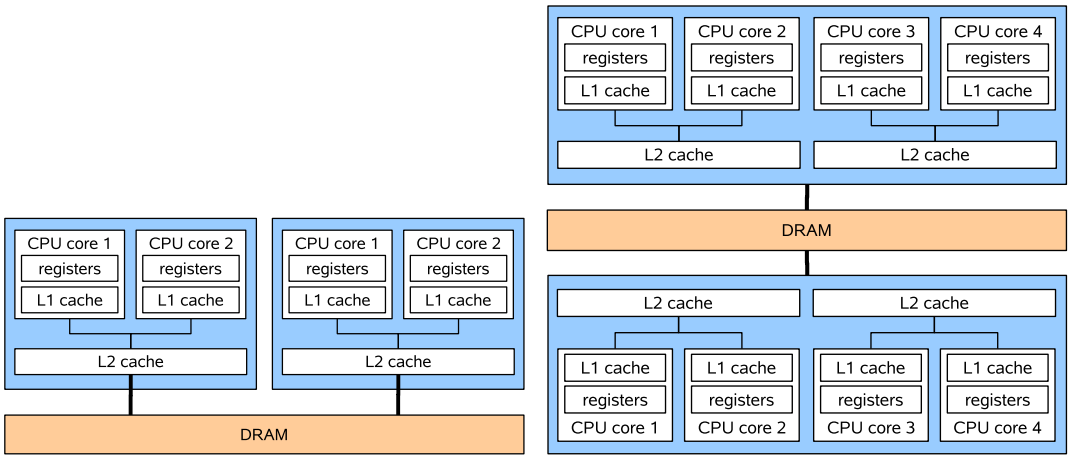
b) Intel Xeon 5160 dualcore processors (codename Woodcrest, cp. Fig. 5.2a), 3 GHz, two of which are grouped in one Dell Power Edge 1950 server. Each processor has a 4 MB L2 cache shared by its cores. This machine has a flat memory model. The Intel 9.1 C++ compiler under Linux was used here.

c) Intel Xeon 5354 quadcore processor (codename Clovertown, cp. Fig. 5.2b), 2.4 GHz, two of which are grouped in a server. Every two cores on one chip share 4 MB of L2 cache. It has to be noted, that I tested on a preproduction version of the processor and the chipset, which might not achieve the full performance of the final version. The Intel 9.1 C++ compiler under Linux was employed here.

High optimization level (`-O3`) and multifile optimization (`-ipo`) were used on all systems.

5.2 Test Cases

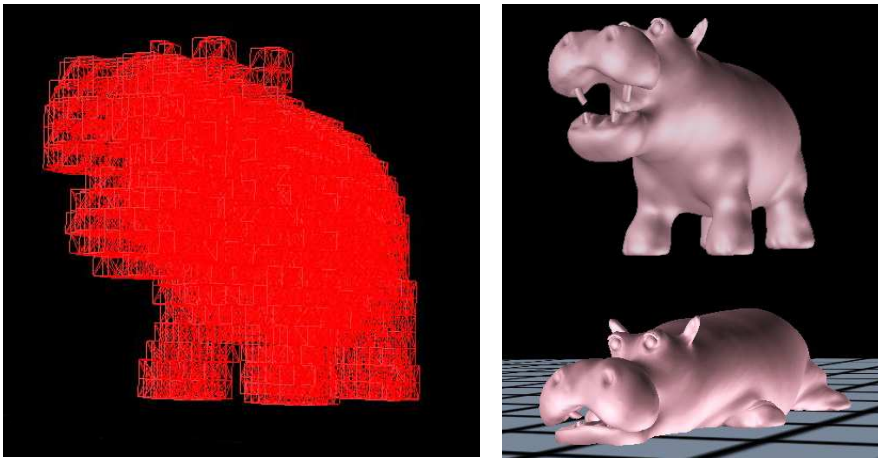
Two benchmark datasets were created. The Hippo dataset (cp. Fig. 5.3) consists of 20,870 tetrahedral elements and 5,550 nodes. The material density is $\rho = 1000\frac{kg}{m^3}$, with an elastic modulus $E = 0.1MPa$ and a Poisson's ratio $\nu = 0.33$. The object



(a) The dualcore Xeon (Woodcrest) system

(b) The quadcore Xeon (Clovertown) system

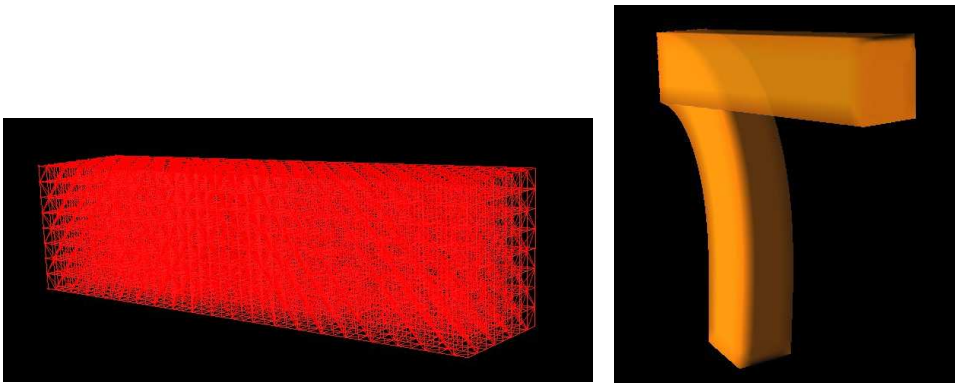
Figure 5.2: The Intel multicore architectures used for testing.



(a) (b)

Figure 5.3: The test case Hippo. The simulation mesh consists of 20,870 finite elements (a).

is falling to the floor without any other constraints. The stiffness matrix remains constant during the simulation and thus the linear FE approach is used.



(a) (b)

Figure 5.4: The test case Bar. The simulation mesh consists of 12,800 finite elements (a).

The second benchmark, the Bar (cp. Fig. 5.4), consists of 12,800 tetrahedral elements and 3,321 nodes. The material density is $\rho = 100\frac{kg}{m^3}$ with an elastic modulus $E = 0.1MPa$ and a Poisson's ratio $\nu = 0.33$. The left side of the object is fixed and the bar is bending under gravity. This is an example of a large deformation, where the stiffness matrix depends on the current deformation state. The corotational FEM is employed to simulate this test case.

Compared to problem sizes that typically require parallelization to be solved in reasonable time, both test cases are rather small, but are still challenging for real time simulation. Depending on the CPU architecture, the matrix and the associated vectors of the Hippo dataset may fit into the on-chip L2 cache. The global stiffness matrix $\mathbb{K}$ has 5550×5550 elements, thereof 63,526 are non-zero (sparse matrix with 0.2% fill rate). Each element of the matrix is a dense 3×3 matrix. The compressed row storage scheme and single floating point precision is used to store the stiffness matrix. As the matrix is symmetric, only the upper triangular matrix is stored. The memory footprint of the global stiffness matrix is approximately 1.3 MB.

Although the global stiffness matrix for the Bar dataset is not built explicitly, for neighbored mesh elements or localized force vectors, it is possible to profit from locality, as will be described later. Thus, the corotational FEM algorithm has a high cache efficiency as well.

5.3 Implementation

This section describes the parallelization of the dynamic FE simulation. First of all, performance analysis experiments were carried out in order to retrieve the runtime profile of both benchmarks without parallelization. An iteration time step of $\Delta t = 40$ ms was used and a total time of 5 s was simulated. Thus, 125 implicit Euler (IE) steps were performed. In each IE step, 10 CG iterations were performed. Table 5.1 shows the portions of total simulation time spent in the most time consuming functions. The major part of simulation time is spent in only two

	Hippo	Bar
`total runtime`	7 s	36.4 s
`ApplyInternalForces`	58 %	11 %
`ModifiedCG`	30 %	88 %

Table 5.1: Runtime profiles of the Hippo and Bar benchmarks. The largest amount of total simulation time is spent in the `ApplyInternalForces` and `ModifiedCG` functions.

functions. The `ApplyInternalForces` function evaluates the internal forces by adding the contributions of all elements to the global force vector (cp. equation (2.32)). The time spent in the `ModifiedCG` function (cp. Fig. 4.15) is dominated by the multiplication of the sparse matrix $\mathbb{A}$ by a vector. For the Hippo, the matrix $\mathbb{A}$ (cp. equation (4.5)) is constant and can be precomputed. It has the same size and sparsity pattern as the global stiffness matrix. However, the matrix-vector multiplication still takes about 60 % of the time spent in `ModifiedCG`. For the Bar, the stiffness matrix (and thus also the matrix $\mathbb{A}$) depends on the current deformation state. The orientation of each element is updated every 200 ms within the `ApplyInternalForces` function. Instead of storing the global stiffness matrix explicitly, the required matrix-vector product is computed on the fly from the contributions from all elements. In this case, the time spent in the matrix-vector multiplication takes about 98 % of the time spent in `ModifiedCG`.

The above analysis shows that the largest benefit can be achieved by the parallelization of both the `ApplyInternalForces` and the `ModifiedCG` methods. The former function contains a loop over all elements summing the forces' contributions into a global vector. The contribution of each element only depends on the current state of the element itself. Typically, this can be done efficiently using a reduction operation. The current version of the OpenMP specification does not allow for reductions on high level data types [TaM06], therefore a private force vector is created for each thread and at the end, all private vectors are summed within a critical section into a shared force vector. This technique is cache efficient, as all updates

```
ApplyInternalForces(const State &state,
                    CVistaPhysicsVector &forces)
{
  #pragma omp parallel shared(state, forces)
  {
    CVistaPhysicsVector priv_forces;
    #pragma omp for nowait schedule(runtime)
    for all elements do
      elem->ApplyInternalForces(state, priv_forces);

    #pragma omp critical
    {
      forces += priv_forces;
    }
  }
}
```

Figure 5.5: `ApplyInternalForces` - parallelized C++ code.

during the loop are written to a local vector that is not distributed among several cores.

Figure 5.5 shows the parallelized code of the `ApplyInternalForces` function. The OpenMP directives are printed in blue bold. The `#pragma omp parallel` construct declares a parallel section. When the program encounters this construct, a team of threads is created to execute the parallel region. The number of OpenMP threads can be either specified as a parameter of this command or in an environment variable `OMP_NUM_THREADS`. The parallel region is executed by all threads unless special directives are used. The `shared` attribute lists variables that are shared by all threads within the parallel region. The `#pragma omp for` construct declares a loop whose iterations will be executed in parallel. The iterations of the loop are distributed among the OpenMP threads that already exist in the parallel region. The `for` loop must have a canonical form. In particular, the number of loop iterations must be known on entry to the loop. The binding between the threads and the loop iterations is controlled by the `schedule` attribute. When `schedule(runtime)` is specified, the scheduling is controlled by an environment variable. The `nowait` clause avoids a synchronization barrier at the end of the `for` loop. The `#pragma omp critical` construct restricts execution of the associated region to a single thread at a time. Within the `ApplyInternalForces` function the critical section is used to avoid concurrent writing when adding the results of each thread stored in a private `priv_forces` vector to the shared `forces` vector. More detailed description of OpenMP directives can be found in [Boa05].

In the `ModifiedCG` algorithm (cp. Fig. 5.6), it is not possible to parallelize the iteration loop, as each iteration depends on the previous one. Each iteration consists of several vector operations, e.g., `Scale`, `Add`, `Dot`, `MultiplyComponents` and

a matrix vector multiplication `Multiply`. Most of these operations can be parallelized in a trivial way. However, a trivial parallelization of the vector operations would lead to a synchronization barrier at the end of each operation. Usually, a

```
ModifiedCG(const CVistaPhysicsSparseSymmetricMatrix &A,
           const CVistaPhysicsVector &b, CVistaPhysicsVector &x,
           const CVistaPhysicsVector &S)
{
  #pragma omp parallel \
    shared(A, x, b, S, r, c, q) \
    private(alpha, beta, delta, delta_old)
  {
    MultiplyComponents(b,S,r);                  // r = constr(b)
    iterate until |r| < epsilon
    {
      delta = Dot(r,r);

      if(first iteration) beta = 0;
      else beta = delta / delta_old;

      delta_old = delta;
      Scale(c,beta);
      Add(c,r);                                 // c = r + beta * c
      Multiply(A,c,q);
      MultiplyComponents(q,S,q);                // q = constr(Ac)
      alpha = delta / Dot(c,q);                 // alpha = rr/cq
      Scale(c,alpha);
      Add(x,c);                                 // x += alpha * c
      Scale(q,-alpha);
      Add(r,q);                                 // r -= alpha * q
    }
  }
}
```

Figure 5.6: `ModifiedCG` - parallelized C++ code. The `Dot` and `Multiply` functions contain an implicit or explicit synchronization barrier (red bold).

CG-type method is parallelized in OpenMP by extending the parallel region over the iteration loop including the system setup. Then the work inside the vector operations can be shared among the threads by using *orphaning*, which allows for placing the worksharing directives in a different scope (e.g., a subroutine) than the enclosing parallel region. Placing orphaned worksharing directives in the subroutines is problematic if these are used in a serial part of the program as well. Therefore, an orphaned version of each vector math subroutine has to be created (cp. Fig. 5.7). The implicit barrier at the end of each `for` loop can be avoided by using the `nowait` clause. However, it is only applicable if the parallelized vector operations ensure that a certain thread is accessing the same indices of the vectors across all such operations. This can be done by using static scheduling of fixed chunk size `N`.

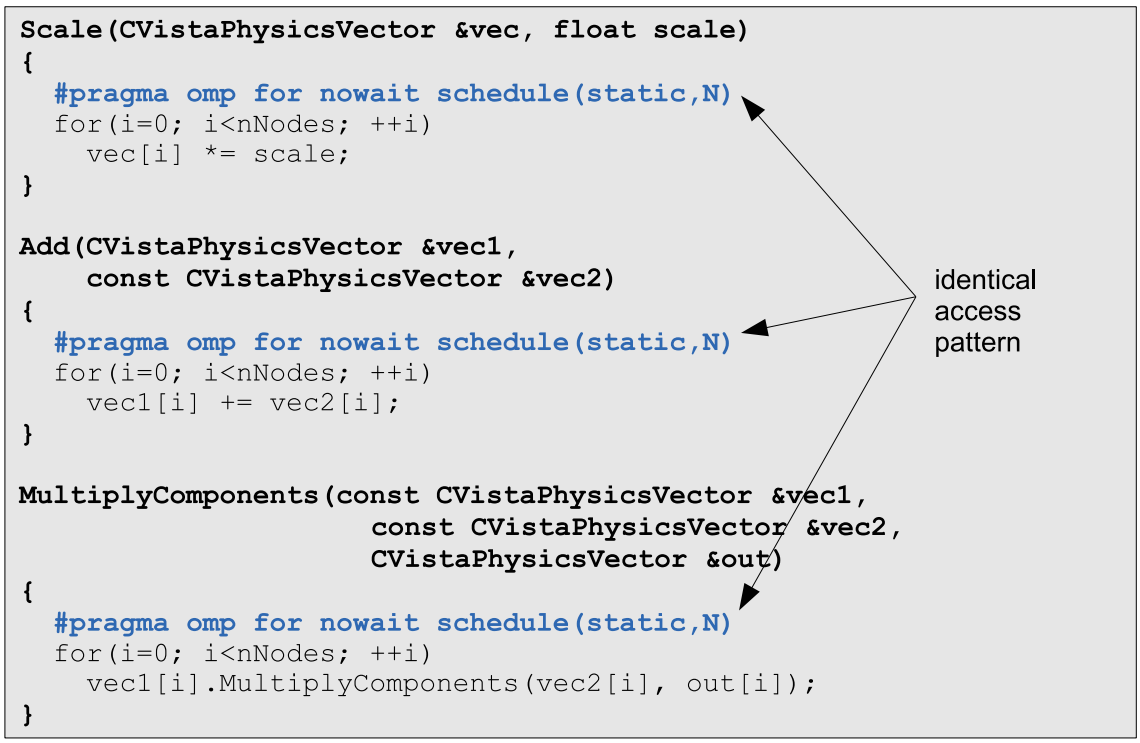
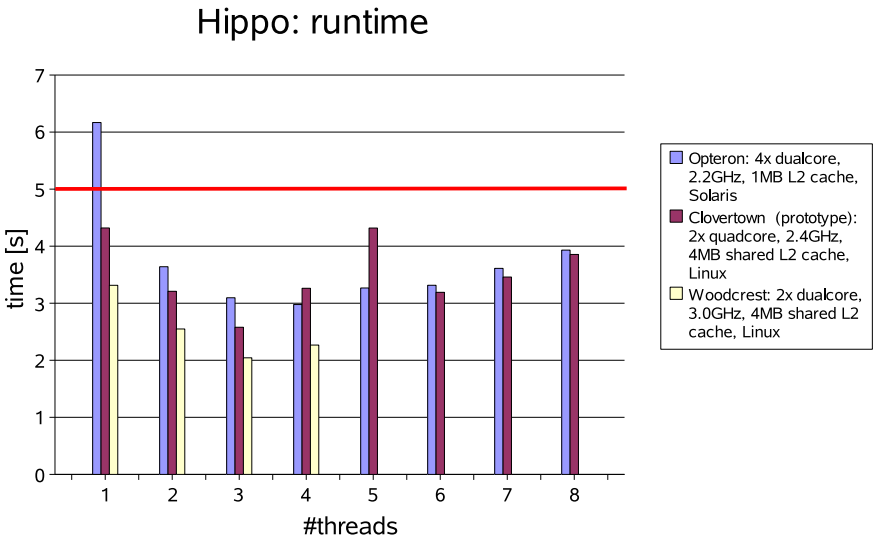


Figure 5.7: Orphaned vector operations. These functions must be called from a parallel region.

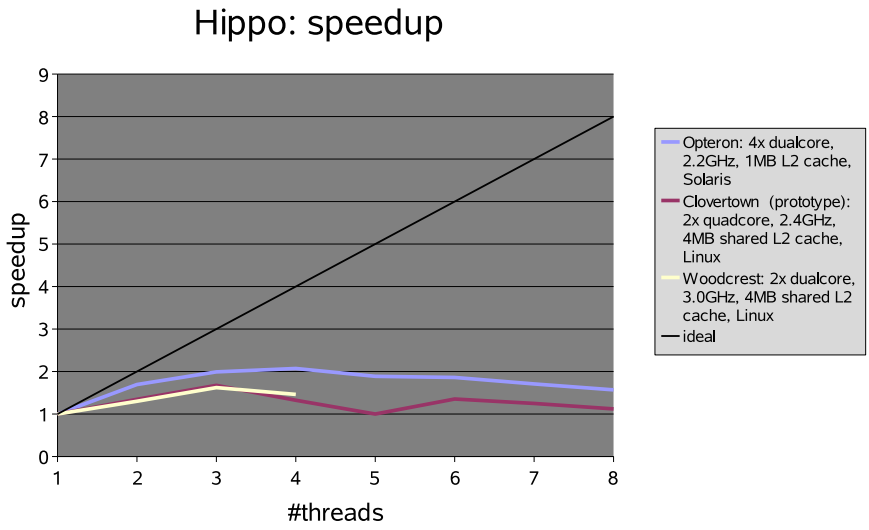
Four synchronization barriers per `ModifiedCG` iteration are needed. The `Dot` function contains an implicit barrier because of reduction. The `Dot` function is called two times per `ModifiedCG` iteration. The `Multiply` function contains an explicit barrier at its beginning ensuring that the input data are ready and an reduction barrier at its end.

5.4 Results

Figure 5.8 shows the results for the Hippo dataset. The simulation scales up to three threads. A speedup of only 1.5-2 can be achieved. The poor scalability is caused by the low computational cost of all parallelized routines compared to the synchronization time. The Sun Analyzer was used to measure the time distribution in the compute kernel and the OpenMP library separately and found the overhead for creating a parallel region and for explicit barriers increasing in the same rate as the compute time decreases. These results were verified by comparing with the EPCC benchmark suite [BO01]. However, the real time limit is reached for over 20,000 elements on one or two cores on all platforms. As the complexity of the algorithm is linear in the number of elements, it is possible to simulate a FE mesh

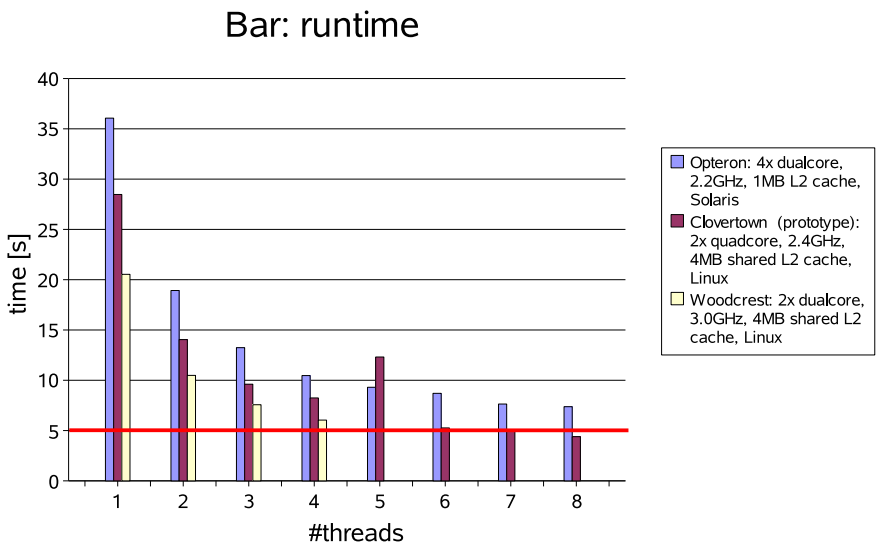


(a) Runtime in seconds for 5 seconds simulation time (red line).

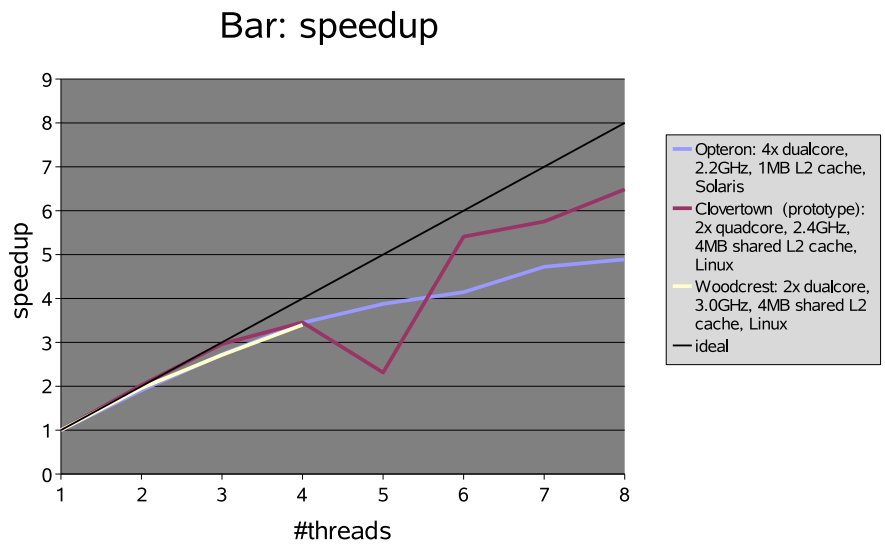


(b) Speedup

Figure 5.8: Results for the Hippo test case. The deformation is computed using the linear FEM.



(a) Runtime in seconds for 5 seconds simulation time (red line).



(b) Speedup

Figure 5.9: Results for the Bar test case. The deformation is computed using the corotational FEM.

with up to approx. 40,000 elements on three Woodcrest or Clovertown cores in real time. For the Clovertown, the required number of cores will already be available in a commodity single socket system.

For the Bar benchmark, the computational costs in both the `ApplyInternalForces` and the `ModifiedCG` functions are higher and, therefore, a better scalability can be expected. Figure 5.9 shows the results for the Bar dataset. A speedup of up to 6.5 was reached on eight Clovertown cores and 3.4 on four cores on all platforms. However, the real time limit for nearly 13,000 elements has only been reached on the Clovertown platform. As the complexity of the algorithm is linear in the number of elements, it is possible to simulate up to 10,500 elements on four Woodcrest cores in real time. This is a noticeable improvement, compared to the approx. 3,000 elements that can be simulated in real time by a serial algorithm.

As can be seen from the results presented above, the computationally most intensive part of an interactive VR application with deformable objects, the FEM based deformation, significantly profits from multicore architectures. The OpenMP API [Boa05] provides a declarative shared model parallelization model while permitting portability. The OpenMP directives extend the C/C++ and Fortran programming languages with SIMD constructs, work-sharing constructs, and synchronization constructs, and they provide support for the sharing and privatization of data. Compared to lower-level parallelization approaches as, e.g., Posix-Threads, OpenMP requires the least design changes of an existing serial code and allows the user to concentrate on the parallelization itself, whereas the implementation of thread creation, management and synchronization is provided by the compiler. The structure of the presented problem, consisting of a number of mutually independent computations of approximately the same runtime meets the abilities of the OpenMP concept. Consequently, the parallelization of the serial algorithms using OpenMP is straight forward, no principal changes to the software design are necessary.

Although the data locality is not managed explicitly, the achieved speedup is comparable to the values achieved by advanced parallelization techniques as domain decomposition and matrix rearrangement (cp. [TPB07]). Moreover, in case of topological changes, the explicit domain decomposition would have to be recomputed, which is a time consuming operation. The main reason why the simple OpenMP approach performs that well, is the small memory footprint of the problem to be solved. As the data needed by each core fits into its cache, rearranging the data with respect to spatial locality does not bring any benefit. The data locality is important for large problems. However, as the speedup grows slower than the number of threads, and in fact, is saturated at certain number of threads, the problem size is limited by the real time requirement.

CHAPTER 6

CONCLUSION

In this thesis I present methods for an interactive simulation of finite elements based deformable objects including topological changes. I present a novel method for interactive cutting of deformable objects in virtual environments. The key to this method is the usage of the extended finite elements method (XFEM). The XFEM can effectively model discontinuities within an FEM mesh without creating new mesh elements and thus minimizing the impact on the performance of the simulation. I show how XFEM is applied to the most common constitutive models used for the interactive simulation of large deformations. Moreover, I present an analysis of mass lumping techniques, showing that the stability of the simulation is guaranteed even when small portions of the material are cut. The XFEM based cutting surpasses the remeshing methods in both, performance and stability and is suitable for interactive VR simulation. The principle of enrichment functions is general and thus can be used with any type of finite elements for creating of a variety of applications involving cutting, cracking or breaking objects.

Further, I designed a software architecture for physical simulation of deformable objects in VR applications. I implemented this architecture and integrated the above cutting method. The framework is suitable for the creation of complex VR applications as, e.g., a surgical simulator. It uses thread level task parallelization for the concurrent execution of visualization, collision detection, haptics and deformation. Moreover, I propose a parallelization approach for the deformation algorithm, which is the most computationally intensive part. The presented solution based on OpenMP requires only minimal changes to the source code while achieving a speedup comparable to the results of more sophisticated approaches. The presented framework benefits from the current developments in the computing industry and allows an optimal utilization of multicore CPUs.

Based on the above framework, I developed a prototype of a surgical trainer and several small testing applications that I used to demonstrate various phenomena throughout this thesis. The current implementation only allows certain cut shapes, an implementation of arbitrary cuts including the extension of an existing cut is desirable. The XFEM is suitable for modeling arbitrary discontinuities, the main challenge here is a consistent remeshing of the geometry surface.

The main differences between minimal invasive and open surgery are the size of the working area and the way the surgeon interacts with the tissue. Generally speaking, the minimal invasive surgery is suitable for smaller interventions, the manipulations with the tissue are simpler, the variability of operating tools is rather limited. In order to create a realistic VR simulation of open surgery, large deformable objects have to be simulated at a reasonable accuracy. A further exploiting of adaptive multiresolution methods in combination with cutting would probably be the good solution to this problem.

In my opinion, the most challenging task is the realistic simulation of open surgery interventions. The surgeons are used to precise bimanual working using a variety of instruments and even their hands and fingertips. The sense of haptics plays a crucial role during an operation. With the current state of both the haptics software and hardware the variety of tools and interventions that can be simulated satisfyingly is strongly limited.

BIBLIOGRAPHY

[AB03] Uri M. Ascher and Eddy Boxerman. On the modified conjugate gradient method in cloth simulation. *Visual Computer*, 19(7-8):526–531, Dec. 2003.

[ACF+07] Jérémie Allard, Stéphane Cotin, François Faure, Pierre-Jean Bensoussan, François Poyer, Christian Duriez, Hervé Delingette, and Laurent Grisoni. SOFA an Open Source Framework for Medical Simulation. In *Medicine Meets Virtual Reality (MMVR'15)*, Long Beach, USA, February 2007.

[AMS02] Fred S. Azar, Dimitris N. Metaxas, and Mitchell D. Schnall. Methods for modeling and predicting mechanical deformations of the breast under external perturbations. *Medical Image Analysis*, 6(1):1–27, 2002.

[AR06] Jérémie Allard and Bruno Raffin. Distributed Physical Based Simulations for Large VR Applications. In *Proceedings of IEEE VR*, March 2006.

[Bat86] Klaus Jürgen Bathe. *Finite Elemente Methoden.* Springer, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1986.

[BB99] Ted Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45(5):601–620, 1999.

[BGTG03] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross. A State Machine for Real-Time Cutting of Tetrahedral Meshes. In

Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, pages 377–386, October 2003.

[BM98] Ivo Babuška and Jens Markus Melenk. The Partition of Unity Method. *International Journal for Numerical Methods in Engineering*, 40(4):727–758, 1998.

[BMG99] Daniel Bielser, Volker A. Maiwald, and Markus H. Gross. Interactive Cuts through 3-Dimensional Soft Tissue. In *Proceedings of the European Association for Computer Graphics 20th Annual Conference. Eurographics'99.*, volume 18/3, pages 31–38, September 1999.

[BO01] J. Mark Bull and Darragh O'Neill. A Microbenchmark Suite for OpenMP 2.0. In *Proceedings of the Third European Workshop on OpenMP (EWOMP'01)*, 2001.

[Boa05] OpenMP Architecture Reviewer Board. OpenMP Application Program Interface, v2.5, 2005.

[BOW+00] Jeffrey Berkley, Peter Oppenheimer, Suzanne Weghorst, Daniel Berg, Gregory Raugi, Dave Haynor, Mark Ganter, and Cole Brooking. Creating Fast Finite Element Models from Medical Images. In *Proceedings of Medicine Meets Virtual Reality*, 2000.

[Bro96] Morten Bro-Nielsen. Surgery Simulation Using fast Finite Elements. In *VBC '96: Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, pages 529–534, London, UK, 1996. Springer-Verlag.

[BSHW07] Cagatay Basdogan, Mert Sedef, Matthias Harders, and Stefan Wesarg. VR-Based Simulators for Training in Minimally Invasive Surgery. *IEEE Computer Graphics and Applications*, 27(2):54–66, March/April 2007.

[BSMM02] Cynthia D. Bruyns, Steven Senger, Anil Menon, and Kevin Montgomery. A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *The Journal of Visualization and Computer Animation*, 13(1):21–42, 2002.

[BV91] William J. Bouma and George Vaněček, Jr. Collision Detection and Analysis in a Physically based Simulation. In *Proceedings of Eurographics Workshop on Animation and Simulation*, pages 191–203, Vienna, 1991.

[BW98] David Baraff and Andrew Witkin. Large Steps in Cloth Simulation. In *Proceedings of Siggraph*, 1998.

[BW99] Jeffrey Berkley and Suzanne Weghorst. Banded Matrix Approach to Finite Element Modeling for Soft Tissue Simulation. *Virtual Reality: Research, Development, and Applications*, 4(3):203–212, 1999.

[BYG94] Ted Belytschko, Y.Y.Lu, and L. Gu. Element-free Galerkin methods. *International Journal for Numerical Methods in Engineering*, 37(2):229–256, June 1994.

[BZXC03] Ted Belytschko, Goangseup Zi, Jingxiao Xu, and Jack Chessa. The extended finite element method for arbitrary discontinuities. *Computational Mechanics - Theory and Practice*, 2003.

[CBM03] Jack Chessa, Ted Belytschko, and Walter P. Murhpy. An Extended Finite Element Method for Two-Phase Fluids. *Journal of Applied Mechanics*, 70(1):10–17, 2003.

[CDA00] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. A hybrid elastic model for real-time cutting, deformations and force feedback for surgery training and simulation. *The Visual Computer*, 16(7):437–452, 2000.

[CGTS04] Murat Cenk Cavusoglu, Tolga G. Göktekin, Frank Tendick, and Shankar Sastry. GiPSi: An Open Source/Open Architecture Software Development Framework for Surgical Simulation. In *Proceedings of Medicine Meets Virtual Reality*, 2004.

[CK00] Hüseyin K. Cakmak and Uwe Kühnapfel. Animation and Simulation Techniques for VR-Training Systems in Endoscopic Surgery. In *Proceedings of the Eurographics Workshop on Animation and Simulation (EGCAS)*, 2000.

[DDBC99] Gilles Debunne, Mathieu Desbrun, Alan H. Barr, and Marie-Paule Cani. Interactive multiresolution animation of deformable models. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, 1999.

[DDCB00] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Adaptive Simulation of Soft Bodies in Real-Time. In *Proceedings of Computer Animation*, 2000.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic Real-Time Deformations using Space and Time Adaptive Sampling. In *Proceedings of the ACM Siggraph*, 2001.

[Del98] Hervé Delingette. Towards Realistic Soft Tissue Modeling in Medical Simulation. Technical report, INRIA France, 1998.

[DKS01] Suvranu De, Jung Kim, and Mandayam A. Srinivasan. A Meshless Numerical Technique for Physically Based Real Time Medical Simulations. In *Proceedings of Medicine Meets Virtual Reality*, 2001.

[DMB+06] Bruce D'Amora, Karen Magerlein, Atman Binstock, Ashwini Nanda, and Bernard Yee. High-performance server systems and the next generation of online games. *IBM Systems Journal*, 45(1):103–118, 2006.

[DMD+00] Christophe Daux, Nicolas Moes, John Dolbow, Natarajan Sukumar, and Ted Belytschko. Arbitrary Branched and Intersecting Cracks with the eXtended Finite Element Method. *International Journal for Numerical Methods in Engineering*, 48:1741–1760, 2000.

[FBD04] Alessandro Faraci, Fernando Bello, and Ara Darzi. Soft Tissue Deformation using a Hierarchical Finite Element Model. In *Proceedings of Medicine Meets Virtual Reality*, 2004.

[FP02] Sarah F. Frisken and Ronald N. Perry. Simple and Efficient Traversal Methods for Quadtrees and Octrees. Technical report, Mitsubishi Electric Research Laboratories, 2002.

[GCMS00] Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Enabling Cuts on Multiresolution Representation. In *CGI '00: Proceedings of the International Conference on Computer Graphics*, page 183, Washington, DC, USA, 2000. IEEE Computer Society.

[GEW05] Joachim Georgii, Florian Echtler, and Rüdiger Westermann. Interactive Simulation of Deformable Bodies on GPUs. In *Simulation and Visualisation 2005*, 2005.

[GFG+97] Sarah Gibson, Christina Fyock, Eric Grimson, Takeo Kanade, Ron Kikinis, Hugh Lauer, Neil McKenzie, Andrew Mor, and Shin Nakajima. Volumetric Object Modeling for Surgical Simulation. Technical report, Mitsubishi Electric Research Laboratories, 1997.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. CHARMS: a simple framework for adaptive simulation. In *Proceedings of the 29th*

annual conference on Computer graphics and interactive techniques, pages 281–290, New York, NY, USA, 2002. ACM Press.

[GLM96] Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of the ACM Siggraph*, 1996.

[GM97] Sarah Gibson and Brian Mirtich. A Survey of Deformable Modeling in Computer Graphics. Technical report, MERL, 1997.

[GQ05] Xiaohu Guo and Hong Qin. Real-time meshless deformation: Collision Detection and Deformable Objects. *Comput. Animat. Virtual Worlds*, 16(3-4):189–200, 2005.

[Hau04] Michael Hauth. *Visual Simulation of Deformable Models*. Phd thesis, University of Tübingen, July 2004.

[HHR+03] Matthias Harders, R. Hutter, A. Rutz, P. Niederer, and Gábor Székely. Comparing a Simplified FEM Approach with the Mass-Spring Model for Surgery Simulation. In *Proceedings of Medicine Meets Virtual Reality*, 2003.

[Hop96] Hugues Hoppe. Progressive Meshes. In *Proceedings of the ACM Siggraph*, 1996.

[HS04] Michael Hauth and Wolfgang Straßer. Corotational Simulation of Deformable Solids. *Journal of the WSCG*, 12(1-3):137–145, 2004.

[Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph.*, 15(3):179–210, 1996.

[ITF04] Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. Invertible finite elements for robust simulation of large deformation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140, New York, NY, USA, 2004. ACM Press.

[JJCK06] Lenka Jeřábková, Jakub Jeřábek, Rostislav Chudoba, and Torsten Kuhlen. Stable Interactive Cutting of Deformable Objects. In *Proceedings of 3rd Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS)*, 2006.

[JJCK07] Lenka Jeřábková, Jakub Jeřábek, Rostislav Chudoba, and Torsten Kuhlen. A Stable Cutting Method for Finite Elements based Virtual

Surgery Simulation. In *Proceedings of Medicine Meets Virtual Reality*, 2007.

[JK07] Lenka Jeřábková and Torsten Kuhlen. Surgical Cutting on a Multimodal Object Representation. In *Proceedings of Bildverarbeitung für die Medizin*, 2007.

[JKT+07] Lenka Jeřábková, Torsten Kuhlen, Christian Terboven, Samuel Sarholz, and Christian Bischof. Exploiting Multicore Architectures for Physically Based Simulation of Deformable Objects in Virtual Environments. In *Proceedings of 4th GI-Workshop on Virtual & Augmented Reality*, 2007.

[JKWP04] Lenka Jeřábková, Torsten Kuhlen, Timm P. Wolter, and Norbert Pallua. A Voxel Based Multiresolution Technique for Soft Tissue Deformation. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 158–161, New York, NY, USA, 2004. ACM Press.

[KB04] Michael Keckeisen and Wolfgang Blochinger. Parallel Implicit Integration for Cloth Animations on Distributed Memory Architectures. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, June 2004.

[KCM99] Uwe Kühnapfel, Hüseyin K. Cakmak, and Heiko Maaß. 3D Modeling for Endoscopic Surgery. In *Proceedings of the IEEE Symposium on Simulation*, 1999.

[Kei06] Richard Keiser. *Meshless Lagrangian Methods for Physics-Based Animations of Solids and Fluids*. PhD thesis, ETH Zürich, 2006.

[KGG96] Erwin Keeve, Sabine Girod, and Bernd Girod. Craniofacial Surgery Simulation. In *Proceedings of 4th International Conference on Visualization in Biomedical Computing VBC'96*, pages 541–546, 1996.

[KGS03] Petr Krysl, Eitan Grinspun, and Peter Schröder. Natural Hierarchical Refinement for Finite Element Methods. *International Journal for Numerical Methods in Engineering*, 56(8):1109–1124, 2003.

[KRS03] Hyun K. Kim, David W. Rattner, and Mandayam A. Srinivasan. The Role of Simulation Fidelity in Laparoscopic Surgical Training. In *Proceedings of Medical Image Computing & Computer Assisted Intervention*, 2003.

[LGB+06] Pablo Lamata, Enrique J. Gómez, Fernando Bello, Roger L. Kneebone, Rajesh Aggarwal, and Félix Lamata. Conceptual Framework for Laparoscopic VR Simulators. *IEEE Computer Graphics and Applications*, 26(6):69–79, November-December 2006.

[LLL+06] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the Performance of 32-Bit Floating Point Arithmetic in Obtaining 64-Bit Accuracy. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing SC '06*, 2006.

[Mal69] Lawrence E. Malvern. *Introduction to the Mechanics of a Continuous Medium*. Prentice-Hall Series in Engineering of the Physical Sciences, 1969.

[MB96] Jens Markus Melenk and Ivo Babuška. The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):289–314, 1996.

[MBB+02] Kevin Montgomery, Cynthia Bruyns, Joel Brown, Stephen Sorkin, Frederic Mazzella, Guillaume Thonier, and Arnaud Tellier. Spring: A General Framework for Collaborative, Real-time Surgical Simulation. In *Proceedings of Medicine Meets Virtual Reality*, 2002.

[MBF04] Neil Molino, Zhaosheng Bao, and Ronald Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics*, 23(3):385–392, August 2004. Special Issue: Proceedings of the 2004 SIGGRAPH Conference.

[MDB99] Nicolas Moes, John Dalbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46:131–150, 1999.

[MDM+02] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable Real-Time Deformations. In *Proceedings of Siggraph*, pages 49 – 54. ACM Press New York, NY, USA, 2002.

[MG04] Matthias Müller and Markus Gross. Interactive Virtual Materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 239–246. Canadian Human-Computer Communications Society, 2004.

[MHB+01] Kevin Montgomery, LeRoy Heinrichs, Cynthia Bruyns, Simon Wildermuth, Christopher J. Hasser, Stephanie Ozenne, and David Bailey.

Surgical simulator for diagnostic and operative hysteroscopy. In *CARS*, pages 79–86, 2001.

[MK00] Andrew B. Mor and Takeo Kanade. Modifying Soft Tissue Models: Progressive Cutting with Minimal New Element Creation. In *Proceedings of Medical Image Computing & Computer Assisted Intervention*, pages 598–607, 2000.

[MRCB06] Thomas Menouillard, Julien Réthoré, Alain Combescure, and Harriddh Bung. Efficient explicit time stepping for the eXtended Finite Element Method (X-FEM). *International Journal for Numerical Methods in Engineering*, April 2006.

[MT03] Matthias Müller and Matthias Teschner. Volumetric Meshes for Real-Time Medical Simulations. In *Proceedings of Bildverarbeitung für die Medizin*, 2003.

[MTCV+04] Nadia Magnenat-Thalmann, Frederic Cordier, Pascal Volino, Michael Keckeisen, Stefan Kimmerle, Reinhard Klein, and Jan Meseth. Simulation of Clothes for Real-time Applications. In *Proceedings of Eurographics, Tutorial 1*, 2004.

[NFP06] Matthieu Nesme, François Faure, and Yohan Payan. Hierarchical Multi-Resolution Finite Element Model for Soft Body Simulation. In *Workshop on Computer Assisted Diagnosis and Surgery*, Santiago de Chile, march 2006.

[NMK+05] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically Based Deformable Models in Computer Graphics. In *Proceedings of Eurographics*, 2005. STAR - State of The Art Report.

[NvdS01] Han-Wen Nienhuys and A. Frank van der Stappen. Supporting cuts and finite element deformation in interactive surgery simulation. Technical report, University of Utrecht, 2001.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical Modeling and Animation of Brittle Fracture. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146, 1999.

[OH05] Kunle Olukotun and Lance Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005.

[OLG+05] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In *Eurographics '05, State of the Art Reports*, pages 21–51, 2005.

[PFTV92] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[Phy06] Ageia PhysX. http://www.ageia.com/physx/, 2006. Last visited on March 28, 2007.

[PKA+05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutre, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. In *Proceedings of Siggraph*, volume 24, pages 957–964, New York, NY, USA, 2005. ACM Press.

[RBST99] Alex Rhomberg, Christian Brechbühler, Gábor Székely, and Gerhard Tröster. A Parallel Architecture for Interactive FEM Computations in a Surgery Simulator. In *Proceedings of the Parallel Computing Conference*, 1999.

[SCMB01] Natarajan Sukumar, David Chopp, Nicolas Moes, and Ted Belytschko. Modeling holes and inclusions by level sets in the extended finite element method. *Computer Methods in Applied Mechanics and Engineering*, 190(46-47):6183–6200, 2001.

[SDF07] Eftychios Sifakis, Kevin G. Der, and Ronald Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 73–80, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[SHGS06] Denis Steinemann, Matthias Harders, Markus Gross, and Gábor Székely. Hybrid Cutting of Deformable Solids. In *Proceedings of IEEE VR*, pages 35–42, Los Alamitos, CA, USA, March 2006. IEEE Computer Society.

[SL05] Herb Sutter and James Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, Sept. 2005.

[SOG06] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. Fast arbitrary splitting of deforming objects. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer ani-*

mation, pages 63–72, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[SP03] Natarajan Sukumar and Jean-Herve Prévost. Modeling quasi-static crack growth with the extended finite element method. Part I: Computer implementation. *International Journal of Solids and Structures*, 40(26):7513–7537, 2003.

[SWT06] Jonas Spillmann, Michael Wagner, and Matthias Teschner. Robust Tetrahedral Meshing of Triangle Soups. In *Proceedings of Vision, Modeling and Visualization (VMV'06)*, 2006.

[TaM06] Christian Terboven and Dieter an Mey. OpenMP and C++. In *Second International Workshop on OpenMP (IWOMP 2006)*, 2006.

[TB06] Bernhard Thomaszewski and Wolfgang Blochinger. Parallel Simulation of Cloth on Distributed Memory Architectures. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'06)*, Braga, Portugal, May 2006.

[THM+03] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of Vision, Modeling, Visualization VMV'03, Munich*, pages 47–54, 2003.

[THMG04] Matthias Teschner, Bruno Heidelberger, Matthias Müller, and Markus Gross. A Versatile and Robust Model for Geometrically Complex Deformable Solids. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 312–319, Washington, DC, USA, 2004. IEEE Computer Society.

[TKH+04] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, Marie-Paul Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Straßer, and Pascal Volino. Collision Detection for Deformable Objects. Eurographics Tutorial, August-September 2004. State of The Art Report.

[TPB07] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Blochinger. Exploiting Parallelism in Physically-Based Simulations on Multi-Core Processor Architectures. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'07)*, 2007.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th*

annual conference on Computer graphics and interactive techniques, pages 205–214, New York, NY, USA, 1987. ACM Press.

[vdB97] Gino van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[VHML99] Gerrit Voß, James K. Hahn, Wolfgang Müller, and Rob Lindemann. Virtual Cutting of Anatomical Structures. In *Proceedings of Medicine Meets Virtual Reality*, pages 381–383, January 1999.

[VVW04] Lara M. Vigneron, Jacques G. Verly, and Simon K. Warfield. Modelling Surgical Cuts, Retractions, and Resections via Extended Finite Element Method. In C. Barillot, D.R. Haynor, and P. Hellier, editors, *Proceedings of Medical Image Computing & Computer Assisted Intervention*, volume 3217 of *LNCS*, pages 311–318, Saint-Malo, France, Sept 2004. Springer Verlag.

[WBG07] Martin Wicke, Mario Botsch, and Markus Gross. A Finite Element Method on Convex Polyhedra. In *Proceedings of Eurographics '07*, 2007.

[WDGT01] Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive Nonlinear Finite Elements for Deformable Body Simulation Using Dynamic Progressive Meshes. In *Proceedings of Eurographics*, volume 20/3, 2001.

[WSH03] Wen Wu, Jian Sun, and Pheng Ann Heng. A Hybrid Condensed Finite Element Model for Interactive 3D Soft Tissue Cutting. In *Proceedings of Medicine Meets Virtual Reality*, 2003.

[ZB03] Goangseup Zi and Ted Belytschko. New crack-tip elements for XFEM and applications to cohesive cracks. *International Journal for Numerical Methods in Engineering*, 57:2221–2240, 2003.

[ZFV02] Florence Zara, François Faure, and Jean-Marc Vincent. Physical cloth simulation on a PC cluster. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 105–112, Aire-la-Ville, Switzerland, 2002. Eurographics Association.

[ZGHD00] Stefan Zachow, Evgeny Gladilin, Hans-Christian Hege, and Peter Deuflhard. Finite-Element Simulation of Soft Tissue Deformation. In *Proceedings of Computer Assisted Radiology and Surgery*, 2000.

[ZTZ05] Olgierd C. Zienkiewicz, Robert L. Taylor, and Jian Zhong Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier Butterworth-Heinemann, sixth edition, 2005.

Curriculum Vitae

Persönliche Daten

Name	Lenka Jeřábková
Geburtsdatum	5. Juli 1976
Geburtsort	Zlín, Tschechische Republik

Schulbildung und Studium

Sept. 1982 - Juli 1984	16. Grundschule in Zlín
Sept. 1984 - Juli 1990	7. Grundschule mit erweitertem Sprachunterricht in Zlín
Sept. 1990 - Mai 1994	Technische Fachschule in Zlín, Studiengang Automatisationstechnik
Mai 1994	Abschluss: Abitur
Okt. 1994 - Feb. 2001	Studium der Informatik an der Tschechischen Technischen Universität in Prag
Feb. 2001	Abschluss: Diplom, Thema der Diplomarbeit: Virtuelles Prototyping von Werkzeugmaschinen
seit März 2001	Promotion an der RWTH Aachen

Arbeitstätigkeiten

Juni 1996 - April 1999	Studentische Hilfskraft in der Firma Multimedia Computer in Prag, Arbeitsbereich: Geographische Informationssysteme (GIS)
Mai 1999 - Sept. 1999	Studentische Hilfskraft in der VR-Gruppe des RZ RWTH Aachen, Arbeitsbereich: VR in Maschinenbau
Okt. 1999 - Juni 2000	Studentische Hilfskraft in der Firma Multimedia Computer in Prag, Arbeitsbereich: GIS, 3D Terrain Visualisierung
Juli 2000 - März 2001	Studentische Hilfskraft in der VR-Gruppe des RZ RWTH Aachen, Arbeitsbereich: VR in Maschinenbau
seit März 2001	Wissenschaftliche Angestellte in der VR-Gruppe des RZ RWTH Aachen

Publikationen

- Lenka Jeřábková, and Torsten Kuhlen. Stable Cutting of Deformable Objects in Virtual Environments using the XFEM. To appear in *IEEE Computer Graphics and Applications*, 2008.

- Lenka Jeřábková, Torsten Kuhlen, Christian Terboven, Samuel Sarholz, and Christian Bischof. Exploiting Multicore Architectures for Physically Based Simulation of Deformable Objects in Virtual Environments. In *Proceedings of 4th GI-Workshop on Virtual & Augmented Reality*, 2007.

- Lenka Jeřábková and Torsten Kuhlen. Surgical Cutting on a Multimodal Object Representation. In *Proceedings of Bildverarbeitung für die Medizin*, 2007.

- Lenka Jeřábková, Jakub Jeřábek, Rostislav Chudoba, and Torsten Kuhlen. A Stable Cutting Method for Finite Elements based Virtual Surgery Simulation. In *Proceedings of Medicine Meets Virtual Reality (MMVR'15)*, 2007.

- Lenka Jeřábková, Jakub Jeřábek, Rostislav Chudoba, and Torsten Kuhlen. Stable Interactive Cutting of Deformable Objects. In *Proceedings of 3rd Work-*

shop in Virtual Reality Interactions and Physical Simulation (VRIPHYS), 2006.

- Torsten Kuhlen, Ingo Assenmacher, and Lenka Jeřábková. Interacting in Virtual Reality. In *K.-F. Kraiss (Ed.): Advanced Man-Machine Interaction - Fundamentals and Implementation.* Springer Verlag, pp. 263-314, 2006.

- Lenka Jeřábková, Timm P. Wolter, Norbert Pallua, and Torsten Kuhlen. Adaptive Soft Tissue Deformation for a Virtual Reality Surgical Trainer. In *Proceedings of Medicine Meets Virtual Reality (MMVR'13)*, 2005.

- Lenka Jeřábková, Torsten Kuhlen, Timm P. Wolter, and Norbert Pallua. A Voxel Based Multiresolution Technique for Soft Tissue Deformation. In *VRST'04: Proceedings of the ACM symposium on Virtual reality software and technology*, 2004.

- Timm P. Wolter, Lenka Jeřábková, Birgit Dickmeis, Torsten Kuhlen, and Norbert Pallua. Virtual Reality as a Training Tool for Plastic Surgeons. In *Proceedings of CARS 2004, Computer Aided Radiology and Surgery*, 2004.