Haptic Dissection of Deformable Objects using Extended Finite Element Method

by

Ziyun Li

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Interactive dissection simulation is an important research topic in the virtual reality (VR) community. There are many efforts on this topic; however, most of them focus on building a realistic simulation system regardless of the cost, and they often require expensive workstations and specialized haptic devices which prevent broader adoption. We show how to build a realistic dissection simulation at an affordable cost, which opens up applications in elementary education for virtual dissections which are currently not feasible.

In this thesis, we present a fast and robust haptic system for interactive dissection simulations of finite elements based deformable objects which supports two type of haptic interactions: point contacts and cuts. We design a semi-progressive virtual dissection scheme of deformable objects in a real-time application. The quality and performance of visual/haptic feedback is demonstrated on a low-end commercial desktop PC with a haptic device.

Acknowledgments

I would like to thank Dr. Jochen Lang and Dr. Emil Petriu for their much-appreciated guidance and support during my research.

Thanks to Lu Sun, Hong Pan and Zheng Xia for their help and support for my thesis. Thanks to Ling Jin for help with rendering and collision detection. Thanks to Kamina and Simon for their always moral inspiration.

This research was funded in parts by the Ontario Research Fund-Research Excellence, Ministry of Economic Development and Innovation and the Natural Sciences and Engineering Research Council (NSERC). We also acknowledge the NSERC NCE, Graphics, Animation and New Media Canada (GRAND) for non-financial support.

Dedication

To my girlfriend Yu Lin, my mother Qiuping Liu, and my family. Without them none of my success would be possible.

Table of Contents

List of Figures

List of Notations

Chapter 1

Introduction

1.1 Motivation

Interactive dissection simulation of deformable objects is an important research topic in the virtual reality (VR) community. An ideal computer-based simulation can provide an inexpensive and repeatable way to setup a virtual environment with highly realistic feedback to the user. It can benefit a broad range of applications such as teaching, medical education, and research. There are already many efforts on dissection simulation of medical procedures [1, 2] and these environments can provide effective safe training environments for high-risk surgical procedures [3, 2]. However, most of them focus on building a realistic surgical simulation, which requires specialized workstations and haptic devices to operate.

Simulation of medical procedures is not our focus; instead we develop a low cost simulation system to make virtual cutting easily adoptable in schools, colleges and universities. We believe that the cost of building such systems is one of the main barriers that prevent them from being used in broader areas that cannot afford expensive equipment at a large scale. With the recent advances in computer hardware and software, a basic consumer PC is able to provide enough computational power for sophisticated real-time simulations. By using the appropriate methods, it is possible to create a realistic dissection simulation of deformable objects at an affordable cost without resorting to ad-hoc or approximate methods. This opens up more opportunities to build an interactive dissection simulation at an affordable cost that might help more professional areas. Frog dissection is one of the main scenarios which our system targets. It is a procedure commonly used in undergraduate courses in zoology and high school biology classes. However, the practice of dissection experiments in education has raised concerns about ethics and health risks. A virtual dissection simulation is a great replacement in such a scenario which offers better repeatability and less health risks. By producing realistic results, it can still deliver good learning experiences to the students.

In such scenarios, the user interacts with the deformable object with two types of instruments: a probe for point contacts ("poking"), and a scalpel for cutting. Using a haptic device with a stylus (pen-like handle) as the human-computer interface, a simulation

Figure 1.1: Sensable Phantom Desktop

is able to provide 3-DOF control for point contacts and 6-DOF control for cuts. Our system uses a Geomagic Touch X (formerly Sensable Phantom Desktop), a 6-DOF point contact device (see Figure 1.1). We choose to work with this device because it is able to render high frequency haptic rendering, and it is available in our lab. However, our system does not require the use of a Sensable Phantom Desktop. Our system use Chai3D [4] as an universal interface to the haptic device so that the user can choose one from a range of commercially available haptic devices that are supported by Chai3D based on circumstances.

There are some key requirements for such an application. A realistic modeling for the object (a frog in this case) is required to simulate the deformation caused by the interaction from the user. In order to deliver good learning experiences, the animation must at least "look right". The performance of a method is also an important factor for an real-time interactive application. The rule of thumb for smooth animation is to keep the update rate above 30 Hz. Last but not the least, the stability and robustness of the simulation is important, especially when a haptic device is used as the input. Unpredictable user inputs with high frequency (500 Hz to 1000 Hz) from a haptic device need to be handled properly in the collision detection between the virtual instrument and the deformable object.

There are many existing methods in the simulation of deformable solids as we review in Chapter 2. Mass-spring systems are arguably the most intuitive way to model deformable objects (see Section 2.1.1). However, its material behavior depends on the resolution and

connectivity of the underlying spring network. Therefore, it requires substantial efforts in parameter tuning to demonstrate a specific material behavior, which make it difficult to setup experiments with different material behavior quickly. Meshfree methods provide another efficient option in modeling deformation. Smoothed Particle Hydrodynamics method (SPH) is one of the popular meshfree methods in this field. It is commonly used in fluid simulation [5], and has been applied in deformable solid modeling [6, 7]. It is a meshfree method that only contains a set of particles and does not have a volumetric representation of the object domain. The dynamics of a particle is approximated by the influences from its neighboring particles. As an approximation method, SPH has poor accuracy in the evaluation of the spatial derivatives of the displacement field in general. Meshfree methods based on the moving least squares approximations (MLS) are a more accurate approximation method, which are used in meshfree solid simulation such as [8, 9, 10, 11]. However, meshfree methods do not naturally have a well-defined boundary of an object. Extra efforts are required in order to generate a surface representation so that we can impose boundary conditions on a deformable object. Although it is still possible to apply boundary conditions, it is not clear how to combine meshfree methods with haptic interaction.

In our research, we have developed a dissection simulation system based on the finite element method (FEM). Compared to other methods mentioned in Chapter 2, the FEM defines a uniform mathematical framework to model solid deformation. It is based on a volumetric representation of the deformable object. A tetrahedral mesh is used as the volumetric representation in our system because it provides great flexibility in modeling the volumetric domain of the object. The tetrahedral mesh provides a well-defined representation of the boundary naturally that can be used to impose boundary conditions, such as surface pressure and displacement constraints. It is very useful in the setup of an biology experiment because the user can setup different postures of the object easily by applying displacement constraints on certain nodes in the tetrahedral mesh. Also, the boundary is used as the user interacts with the object. For example, "poking" on the object can be translated into applying pressure on the surface of the deformable object by the tip of the haptic device. Since the FEM is able to capture the material behavior and the volumetric behavior of the deformable object correctly, a deformable object modeled by the FEM is volume conserving while the user applies pressure on its surface. A well-defined boundary of a deformable object also simplifies the implementation of non-invasive operations, which are common in a biology experiment. Because the FEM is based on elasticity theory, it is possible to specify the material properties of an object quantitatively and avoid tedious parameter tuning. This is much preferred in our scenarios in order to convey the correct intuitions to students.

However, the topological encoding from the tetrahedral mesh also limits the flexibility to model dissections on top of the FEM. In a cutting scenario, we need to model the discontinuities created by the user. There are many efforts to model cuts on the FEM as we review in Section 2.3.2. However, they in general require substantial modifications of the topology of a deformable object as a cut proceeds [12, 13, 14, 15], which are not preferable in a real-time simulation. Our system adapts the extended FEM (XFEM) [16, 17] which can create discontinuities by introducing a global discontinuous function to the solution space. It is not required to remesh the volumetric representation of the object in order

to model the cuts. It only causes small increases in the numbers of DOF around the intersected region to model the deformations of the separated parts. And as we discuss in Section 3.3, the additional DOFs are appended to the original system without modifying its internal structure. Modifications of the original system with high computational cost are therefore avoided, which makes the XFEM well suited for interactive applications.

1.2 Thesis Statement

It is possible to build a dissection simulation with realistic visual and haptic interaction with affordable computation power.

Using available open source libraries, we are able to reuse some components in our implementation of the dissection framework. The simulator based on our framework is able to achieve stable real-time performance of a cutting simulation on a normal desktop PC. By using a haptic device as the interface, users can also interact with the virtual environment, which greatly increases the sense of immersion. Our framework can provide stable semi-progressive cutting computationally inexpensively while simulating appropriate physical properties of the deformable objects. We believe that our framework can give rise to more applications of physically-based simulations with affordable budgets.

1.3 Major Contributions

In this thesis, we present an extensible haptic system for interactive dissection of finite elements based, deformable object simulation which supports two type of haptic interactions, i.e. point contacts and cuts. The main objective is to create a fast and robust interactive cutting simulation which allows users to visualize, feel and interact with the virtual environment with affordable costs. The main contributions are:

- **Design a semi-progressive cutting scheme of deformable objects**.

  Our framework adapts the extended finite element method (XFEM) to describe the discontinuities without changing the underlying volumetric mesh during the simulation. This alleviates the performance cost caused by changing the topology of the volumetric mesh and improves the stability of the simulation. Instead of using predefined planes, we use the sweeping plane of the cutting front between consecutive time-steps to create cuts on the deformable object. A stable volume collision detection method is developed to find the exterior and interior elements that interact with the sweeping plane of the cutting front. As a result, users are able to dissect the deformable object semi-progressively and observe the dynamic change of the deformable object as the cutting front proceeds with limited computational power.

- **Demonstrate the feasibility of real-time dissection simulation in a computationally inexpensive way**.

We present a real-time dissection simulation with the support of two typical type of haptic interactions: point-like contacts and cuts. Our system leverages several open source libraries including Chai3D [4], OpenVolumeMesh [18], Bullet [19] and Vega FEM [20]. By using existing open source libraries, we avoid the cost of reinventing the wheel and focus on implementing the missing parts in the creation of an interactive dissection simulation. We outline a general work flow to build a simulation system that is robust to inconsistent haptic input and capable of producing a convincing visual appearance of the deformable object. We evaluate the performance of our system on a commercial desktop PC.

1.4 Thesis Overview

In this thesis, we present an extensible haptic system for interactive virtual dissection of a finite element based deformable object simulation. Our system doesn't use any proprietary software except for the drivers of the haptic devices, and is constructed with several open source libraries. We design an abstraction layer on top of our implementation of the deformable object and the haptic tool. The interface-based design does not imply any assumption on the underlying data, such that it would be possible to replace some components with other libraries without changing the application code. Our system doesn't depend on any specific hardware either. The framework of our system is designed to be portable which facilitates creating a similar system on different operating systems and with various off-the-shelf haptic devices. We believe the system can serve as a stepping stone to further haptic research.

The core feature of our system is its support for haptic dissection of deformable objects. We show how to build a stable cutting simulation on an off-the-shelve consumer computer system. The input of our system is a tetrahedral mesh and several physical properties of the deformable object. The system is responsible to interface a haptic device, visualize the object, resolve collision between the object and the cutting tool in the virtual environment, and provide haptic force feedback. Our implementation is tested on a consumer desktop PC equipped with a Geomagic Touch X (formerly Sensable Phantom Desktop) (Figure. 1.2). It is worth noting that our system does not require the use of a Sensable Phantom Desktop, which is expensive compared to other economic options. By using Chai3D [4], our system is also compatible with low-end haptic devices such as Novint Falcon and Geomagic Touch (formerly Sensable Phantom Omni). However, since we have direct access to a Sensable Phantom Desktop in our lab, we keep using it as the main haptic interface for our system.

We present a high-level overview of our system in Figure 1.3. The core of our system is based on the finite element method that simulates the dynamic behavior of deformable objects. Our system takes a tetrahedral mesh, which defines the shape and physical properties of the deformable object, as the input. Based on the information provided, the system computes constant properties of the deformable object such as the stiffness matrix and the mass matrix before the simulation. Also, the surface mesh that is used for rendering and collision detection is generated from the tetrahedral mesh. Before entering the simulation loop, the system initializes all necessary components including the deformable

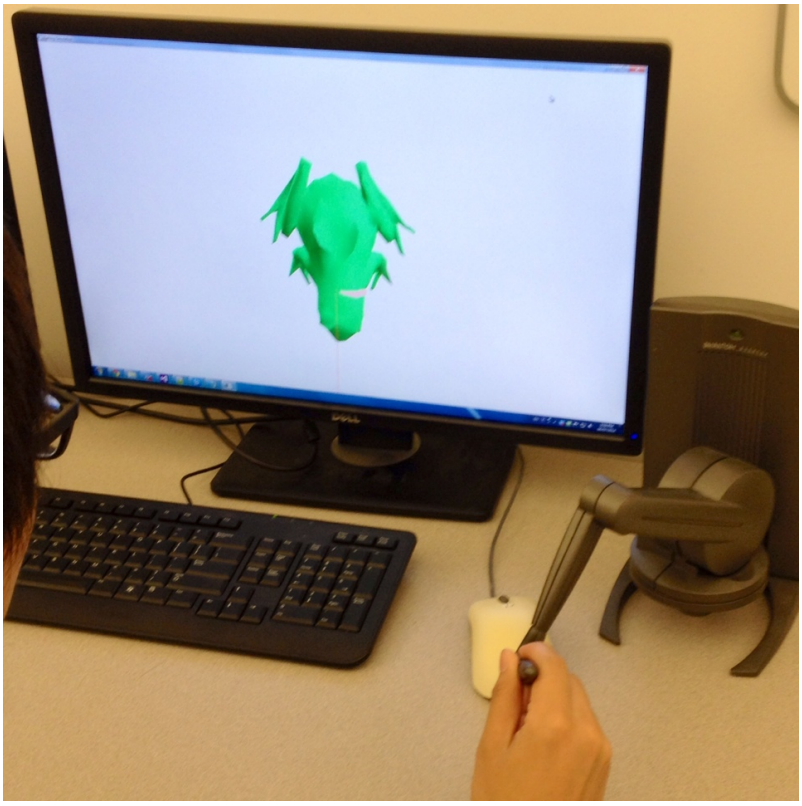


Figure 1.2: A User Cutting a Frog in the Scene

object and the haptic device. Two separate threads are used to process the deformable object dynamics and the haptic interaction, respectively. The dynamic thread is mainly responsible to compute the displacements when the deformable object needs to react to the external force or a cut. The haptic thread is responsible to resolve collision between the haptic tool and the deformable object, and for rendering the force feedback to the haptic device if a haptic device is available. Notes that these two threads are operating at different refresh rates. Since the displacement of the deformable object is also needed for visualization, we expect the frequency of the dynamics thread to be above 30 Hz to guarantee visually smooth rendering. The haptic thread requires much higher update rate which ranges from 500Hz to 1000Hz, so the system adapts a computationally light-weight way to resolve collision and compute the contact force, which is covered in Section 4.4.

Our research was inspired by related work in the field of physically based deformable models and haptic rendering. In Chapter 2, we discuss related research done in these fields. In Chapter 3, we give a brief background about the physics and computation methods underlying our system. Basic continuum mechanics and the linear finite element method (FEM) is discussed. In order to achieve stable and efficient dissections, we adapt the extended finite element method (XFEM) in our system, which will be covered in the same chapter. In Chapter 4, we present the high-level design of our dissection simulation. The actual degian and the implementation details are presented in Chapter 5 The performance are evaluated based on experiments in Chapter 6. The final Chapter 7 concludes our presentation and discusses some future work based on our research.

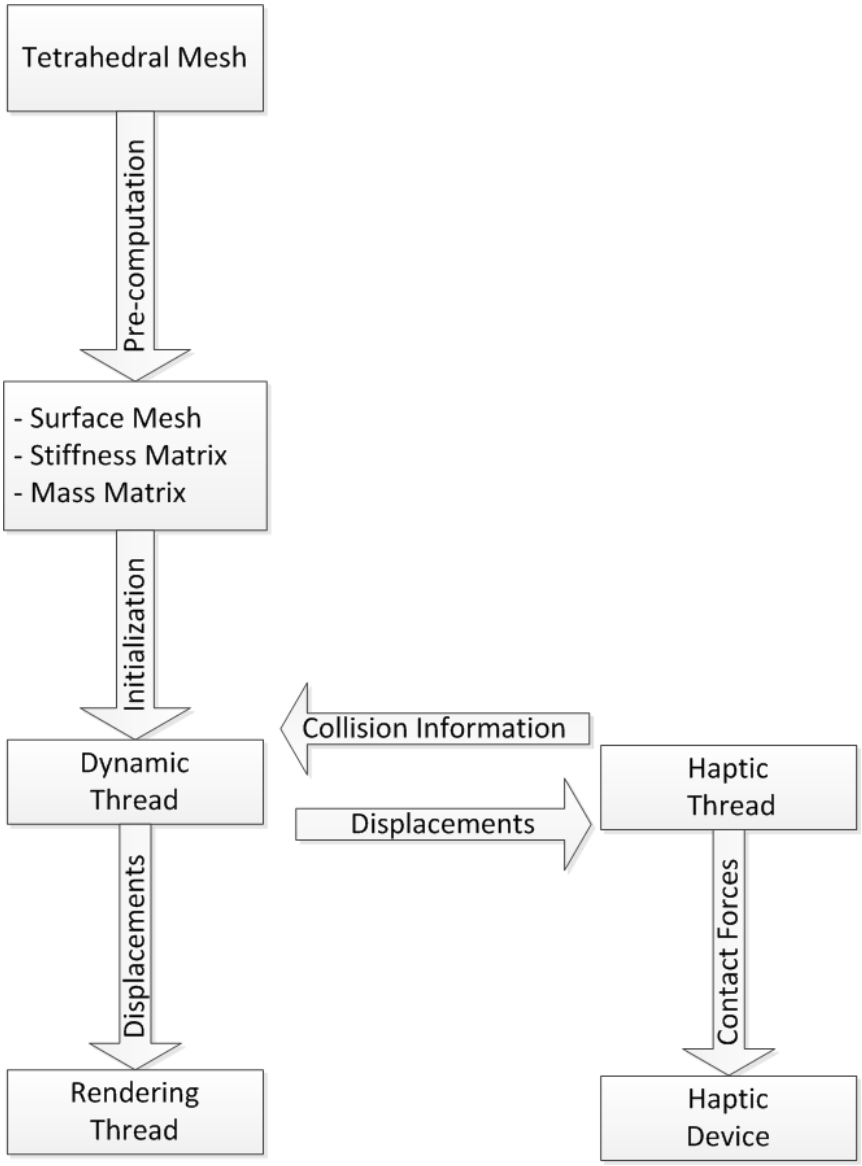


Figure 1.3: High-level System Overview

Chapter 2

Related Work

Our system builds on various research of physically based deformable models and haptic rendering. Section 2.1 and 2.2 gives a brief introduction of the existing methods to model deformable objects. We review different deformable models in the computer graphics community and discuss their strengths and weaknesses. We focus more on the physically-based methods in Section 2.2. Section 2.3 specifically discusses the virtual dissection simulation based on different deformable models. Section 2.4 discusses related work of haptic rendering in interactive virtual environments.

2.1 Non-Physically-Based Deformable Models

2.1.1 Mass-Spring System

Mass-spring systems are arguably the most intuitive way to model deformable objects As the name implies, mass-spring systems typically consist of point masses connected by a network of springs. Because of its simplicity, it is easy to implement and generally achieves good computational efficiency in most scenario (e.g. non-stiff systems).

The applications of mass-spring system have been researched extensively in cloth simulation. Breen *et al.* [21] present a particle system to model cloth. By using real-world measured data, they produced highly realistic images of static drape. However, their method focuses on simulating the static behavior of cloth, which makes it inappropriate for animation or real-time applications. Provot [22] presents an improved mass-spring system that can handle the "super-elastic" problem, which happens when a large stress concentrates on a small region of the cloth surface (e.g. the hanging points of a flag). Although his heuristic methods achieve realistic results, there is no physical justification for them. Baraff *et al.* [23] propose an implicit integrator method to avoid numerical instability caused by stiff equations in cloth simulation. We refer the readers to a more extensive overview of the cloth simulation such as [24] and [25].

Desbrun *et al.* [26] present a stable and efficient method to animate mass-spring systems. Their integrator scheme is derived from an implicit integration method, and in order

to improve the performance of solving the system at each time step, a predictor-corrector approach is used. In this approach, an approximation of the solution of implicit integrator is computed first; then they correct the estimate in a post processing step to guarantee momentum preservation. An inverse kinematics process is also included to deal with constraints to the system, e.g. collision.

The main drawback of mass-spring systems is the fact that they are not built upon elastic theory. As a result, the behavior of a mass-spring system depends on the resolution and connectivity of the spring network, and therefore requires extra efforts in tuning the parameters which do not usually have good physical interpretation. Also, the mass-spring systems cannot capture volumetric effects, such as volume conservation, which makes it inappropriate to model deformable solids. As a result, mass-spring systems have applications in simulating cloth, because cloth can be viewed as not having any volume. Inhomogeneous cloth can be successfully simulated by carefully arranging the structure of the spring network and tuning the spring stiffness.

For some applications, such as animation or video games, these systems still produce visually realistic results; however, in terms of solid simulation, a physically based method is preferred in order to correctly demonstrate the material properties and volumetric effects of the simulated deformable object. As we mention in Chapter1, the user of our system should be able to "poke" the deformable object, in which case the volumetric effects of the object is important. And we also want to avoid tedious parameter tuning when setting up the simulation.

2.1.2 Shape Matching

Müller *et al.* [8] propose a meshfree method for simulating deformable objects based on geometric intuition. The deformable object is modeled by a set of sample points. At each time step, their method computes the least square optimal linear transformation between the reference and the current configuration of the point set. And the goal positions of the point set in the next time step can be found using the rotation component of the linear transformation. Their method can also model linear deformations (i.e. shearing and stretching) and quadratic deformations (i.e. twisting and bending). They prove that their explicit integration scheme is stable as long as the external forces are either a constant force field or applied instantaneously. However, if the particles are closed to coplanar or collinear, the linear transformation becomes ill-conditioned or even singular, causing the simulation to be unstable.

Müller and Chentanez [27] extend the shape matching method by incorporating the generalization of the position-based dynamics (PBD) [28]. In their method, each particle in the system is attached with the orientation information in addition to the position. In order to deal with the coplanar and collinear configuration, they define and compute the moment matrix for a particle by using its orientation information. Their integration scheme is based on PBD and leverages the orientation information of a particle. However, the representation of a deformable object is formed by a set of particles and a set of edges

connecting them. Since the topology of the object is introduced, it limits the flexibility to realize some effects that modify the topology of the object.

As a meshfree particle system, shape matching methods does not naturally have a well-defined boundary, which is important for imposing boundary conditions. While it is possible to impose boundary conditions in a meshfree method, it is not clear how to incorporate the haptic interaction.

2.1.3 Heuristic Methods

Also, there are some intersecting heuristic methods to model deformations. Gibson [29] propose the 3D ChainMail approach which applies motion constraints on the volume elements in the deformable object. The deformation of a displaced element is either relatively small that only influence the element locally, or excess the constraint limit so that the deformation is propagated to its neighboring. After the geometric changes, they apply an energy relaxation algorithm so that the volume element's position is adjusted to reduce the energy of the object. The 3D ChainMail method only performs simple computations during the simulation, and is able to model a wide range of dynamic behaviors of the objects.

Conti *et al.* [30] model a deformable object by filling spheres inside the volume occupied by it. Those spheres are placed along the medial axis transform of the object, and they are connected by a set of springs. This essentially forms a skeleton of the deformable object. The surface mesh and the skeleton are also connected by a set of springs that control volume preservation and transfer forces between each other. Their method is very computationally efficient and visually plausible.

However, it is hard to establish a quantitative connection between the parameters and the real-world physical properties for a heuristic method. Proper physical interpretation for the parameters is preferred in an educational scenario in order to convey the correct intuition to the students. Therefore, a physically-based model is more suitable to our system.

2.2 Physically-Based Deformable Models

Physically-based deformable models have been in the computer graphics community for more than three decades. Much research in these fields have been used in a wide range of off-line and real-time applications, such as surgical planning, surgical simulations, and video games. The types of deformable objects that can be successfully modeled are not limited to solids; the simulations of other continua, such as fluids, gases, and smoke, have also been applied in the film industry. However, the discussion of simulation methods for fluids and gases is beyond the scope of this thesis. In this section, we discuss state-of-the-art techniques in the simulation of deformable solids.

We also refer the interested readers to the following overviews for a broader summary of recent work in the field of physically based modeling. There is an overview by Nealean *et*

al. [31] that covers the existing physically based models of deformable objects in computer graphics. A survey by Meier *et al.* [32] discusses deformable models in the context of real-time surgical simulation. Wu *et al.* [33] presents a very recent survey of the physically-based virtual dissection simulation in deformable bodies.

The existing physically-based models can be roughly categorized into two types: (1) mesh-based methods in which the deformable object is modeled by a geometrical and topological representation (i.e. volumetric mesh), and (2) meshfree methods in which the deformable object is modeled by a set of particles without an explicit topological representation. Each category has advantages and disadvantages. We discuss both of them in the context of deformable solid simulation.

Mesh-based methods are based on a volumetric mesh representation of the deformable object. The volumetric mesh is then the simulation grid of the material which defines the geometry and the topology of the deformable object. There are various representations of the volumetric mesh that have been used by works in this field, including tetrahedral meshes, hexahedral meshes and polyhedral meshes. The volumetric mesh also provides a well-defined representation of the boundary naturally that can be used to impose boundary conditions. However, since the topology is encoded in the volumetric mesh, it is not flexible enough in the scenarios where the connectivity of the volumetric mesh needs to be changed, such as virtual cutting.

In contract to mesh-based methods, meshfree methods do not require a simulation grid; instead, they are based on particles. There is no explicit encoding of the material topology in these methods; therefore it is more flexible in the scenarios where the connectivity of the deformable objects varies during the simulation. Therefore, meshfree methods are used extensively in fluid and smoke simulations. However, these methods are required to maintain and update the adjacency information of the particles in every simulation step. The neighbor search method used to update this information is essential to the performance of meshfree methods. Also, there is no well-defined boundary in meshfree methods, which requires additional efforts to create the surface of the deformable object during the simulation.

2.2.1 Finite Element Method

The finite element method (FEM) is a more physically precise method to model a deformable object compared to mass-spring systems. Beyond its use in the field of real-time graphics and animation, it is one of the most common methods in the analysis of structural mechanics. In the framework of FEM, the deformable object is viewed as a continuum, meaning the matter in the body is continuously distributed to the space it occupies. In order to solve the system, the object is divided into a collection of sub-domains, which are often referred to as *elements*. Each element is represented by a set of equations that uniquely describes its state of stress, strain and displacement, and the whole system which is combined by all sets of equations is solved for the final solution.

A lot of work in the computer graphics community uses FEM to simulate deformable objects. Unlike other disciplines, the applications of FEM in computer graphics emphasize

performance rather than physical accuracy. In order to support real-time applications, previous research often uses a simple form of FEM. For example, almost all methods based on tetrahedral meshes use a linear shape function as the interpolation scheme to reconstruct the deformation map from discrete sample points. Also, explicit FEM is popularly used which computes the nodal forces locally from adjacent elements instead of iteratively solving it to enforce the equilibrium of the internal structure forces with the externally applied load. While the simple form FEM might not be suitable for engineering analysis, it is able to achieve visually realistic results without much computational burden.

Linear FEM

Cotin *et al.* [34] propose a linear FEM based method with real-time interactive update rate. The deformation of the object is driven by the displacement constrains instead of force. Then the contact force on the haptic tool and the surface of the object is computed from the global deformation. They pre-compute a set of equilibrium states of element deformations, and then use the results in the simulation to speed up the performance. Also, haptic force feedback and collision detection are included in their system. But the pre-processing of their method limits the flexibility of the simulation, which makes it hard to change the topology of the underlying deformable objects during the simulation.

Generally speaking, in order to model deformations, we need to solve a set of partial differential equations (PDE) which governs the dynamics of the deformable object, which is essentially a boundary value problem (BVP). While FEM is a popular technique to find the approximations of a BVP, the boundary element method (BEM) is an interesting alternative method to FEM. Instead of on the volume of deformable objects, all computations are done on the surface (boundary), and it essentially reduces a three-dimensional problem to a two-dimensional one. James and Pai [35] develop their ArtDefo system which can achieve real-time simulation of deformable objects using a BEM with linear elasticity. Because of the linearity of the model, a set of reference boundary value problems (RBVPs) can be pre-computed and used in real-time simulation. However, their method is based on the assumptions that the underlying material of the deformable object is homogeneous; therefore it cannot support inhomogeneous materials, which limits its application. Also, it is based on the linear elasticity which makes it inappropriate in a large deformation scenario.

One important feature of the linear FEM approach is that the stiffness matrix of the system is constant and diagonally dominant. As a result, simulations using a linear FEM are generally fast enough for real-time applications. However, the small strain tensor used in linear FEM makes it only valid for very small deformations. In a simulation of deformable objects, very large deformations can occur because of many reasons, e.g. compression, twisting, and etc. In these cases, a linear FEM is unable to generate visually realistic results. Also, a rotational motion without any shape change would generally produce a non-zero strain and eventually a non-zero stress. As a result, in a large deformation scenario, objects increase inaccurately in volume and are not visually realistic. While linear FEM has the efficiency we look for, it is not suitable to convey correct intuitions to the students because of the visual artifacts in rotation scenarios.

Non-Linear FEM

Because of the inaccuracy of the linear FEM in a large deformation scenario, many researchers try to adapt a non-linear FEM approach in the simulations of deformable objects. Zhuang and Canny [36] present a real-time simulation system of deformable objects with global deformations (i.e. large twisting and bending that involves the entire body of an object) using a non-linear FEM model. They use a quadratic strain tensor that handles rigid body motions correctly. Since the elastic force is no longer linear to the displacement, the explicit Newmark integration scheme is used to avoid solving the nonlinear system. By restricting time steps to a small set of values, only a small number of possible matrices need to be inverted. This fact enables the pre-computation of all the inverse matrices, which are represented by their LU factorization. With the pre-computed information, at each step only back-substitution is needed and a majority of the computational work is done beforehand.

Debunne *et al.* [37] present an adaptive method for animating visco-elastic deformable objects in real-time applications. Their method adapts a non-linear FEM model to deal with large deformation scenarios. To alleviate the computational cost in solving such a non-linear system, a space and time adaptive level of detail (LOD) technique is used to locally adapt the sampling according to the current local deformation. The deformable object is modeled as a multi-resolution hierarchy of tetrahedral meshes. By using a quality criterion, their method can find the volume where the simulation is too coarse and then replace it with a finer sampling. By choosing an appropriate criterion, it essentially concentrates the computational resource to the parts of the deformable object that deform most.

Wu *et al.* [38] also propose an adaptive method which refines the volumetric mesh during the simulation. They introduce the concept of dynamic progressive meshes to provide sufficient deformation details while avoiding unnecessary computations. The deformable object is modeled by a hierarchical mesh structure which is precomputed offline for each level of the hierarchy. The finest mesh is coarsened through edge collapses, and the information of this process is stored in a hierarchical tree. During the simulation, an error estimator determines where local mesh refinement is necessary. They consider both material and geometric nonlinearities in their simulation system. To compensate the computational cost induced by non-linear elasticity models, their method use both mass-lumping and an explicit integration scheme.

Irving *et al.* [39] present a method to robustly handle arbitrary large deformation and inverted finite elements that is general with any material constitutive model by computing the correct diagonalization of the deformation gradient. Essentially, by using the singular value decomposition of the deformation gradient, their method extracts two rotations that rotate the world and the material spaces respectively to a space where the deformation gradient is diagonal matrix. The deformable object is modeled by tetrahedral meshes with linear shape functions. Their method is general enough to be applied to different non-linear constitutive models to avoid element inversion issues.

The main problem of non-linear elasticity is that the stiffness matrix is no longer constant. For implicit integration, the stiffness matrix must be re-computed at every time

step, which is very computationally expensive. As a result, most of the methods based on non-linear elasticity are integrated explicitly. In order to guarantee the solutions of such systems are stable, sufficiently small time steps are required, which also increases the computational cost. Therefore, these methods are usually used to model non-stiff objects, which do not have very serious instability issues during dynamic simulations. However, we want to build a simulation system that is general to most common scenarios. A system that is only able to model soft objects limits the range of the deformation behaviors it can support.

Co-rotational Linear FEM

Müller *et al.* [40] [41] introduce the co-rotational concept for the linear FEM. Essentially, the idea is to decompose deformation into the rigid-body motion and strain motion. In real-world scenarios, the rigid-body motion should not cause the change of the strain of the deformable object. However, the small strain tensor used in linear FEM is not rotationally invariant, making it only valid for very small deformations. The co-rotational method overcomes the weakness of linear FEM in a large deformation scenario by removing the rotation information of the deformation.

In [40], they propose a stiffness warping method that extracts the rotational component of each node. The global stiffness matrix does not need to be assembled at each step, but ghost forces are introduced because the forces in their model are not guaranteed to yield zero total momentum as elastic forces should. In [41], they improve the co-rotational method by extracting the rotational component of the deformation for each element Then the deformation forces are computed aligned with the reference configuration. This method produces stable and more visually realistic results.

Parker and O'Brien [42] present the design of a physics engine based on the FEM which is suitable for video gaming. The objects in the scene are modeled by tetrahedral meshes. In order to avoid visual artifacts in large deformation scenarios, their system adapts the co-rotational linear FEM [41] to model the deformable object. They describe in detail the design of their system, and outline a general solution of implementing such systems that is robust to unpredictable user interactions with reasonable real-time performance. It demonstrates the feasibility of FEM in a highly dynamic real-time application, even though it has a notorious reputation of the computational cost.

We find that the co-rotational FEM meets our requirements for a deformable model. The resulting system of the object is linear, therefore it is possible to use implicit integrator to support a wide range of stiffness and timestep. The visual artifacts from the linear FEM are also avoided. Although the FEM has a notorious reputation in the computational costs, Parker and O'Brien [42] shows that a FEM-based simulation is still to real-time applications.

Model Reduction Methods

Bro-Nielsen *et al.* [43] develop a real-time surgical simulation based on linear FEM. In order to achieve real-time performance (i.e. 20 frames/second), they propose a condensation method that compresses the linear system of the volumetric domain to a system which has the same degree of freedom as a surface model. Although the size of the system is reduced remarkably, the system is able to capture the volumetric deformation of the object. The resulting stiffness matrix of the deformable object after the condensation is dense. They explicitly invert the dense system before the simulation to allow fast multiplication against vectors so that the system is solved efficiently at run time. However, since the dense system only contains the surface nodes, it is not able to model cuts that involve the interior nodes without the full system of the object.

Barbič and James [44] present an efficient approach to model geometrically non-linear deformation using model reduction method. The method is based on the fact that large deformation models with linear materials result in internal force models that are simply cubic polynomials in reduced coordinates. Therefore, the implicit integration is performed in reduced coordinates and therefore independent of the geometric complexity. Since the computational cost of the implicit integration is greatly reduced, they are able to use non-linear deformable object with an implicit integration scheme, which is considered to computationally expensive for real-time applications. They also employ GPU to compute matrix-vector multiplication to improve the performance. Their system achieves great real-time performance for non-trivial meshes and is able to support haptic rendering.

Huang *et al.* [45] propose an subspace method to improve the efficiency and stability in modeling deformation with non-linear constraints. In their method, a coarse mesh is built from the original finer mesh and then used as a control mesh. Detail preservation is expressed by the deformation energy casted on the control mesh. Their method is able to produce interactive animation of the deformation. However, the precision of their method is not easy to analyze mathematically. By using a multi-resolution method, their system can use a coarse mesh with 200 vertices and a fine mesh with 30000 vertices to simulate the detailed deformation of the original mesh with 170000 vertices efficiently.

Nesme *et al.* [46] propose a composite element method using model reduction. They use a coarse mesh to simulation the deformable of an object. However, their method takes into account the topological details and the varying material properties within a coarse element. Since the topology of the model is preserved, their method is able to model complex systems with branches and holes even the simulation is performed in a lower-order domain. Therefore, detailed deformation of the high resolution surface can be modeled faithful by the composite elements.

The model reduction methods above focus more on improving the efficiency of deformation, and it is not clear how to incorporate dissections with their reduced models. We review the work of Niroomandi *et al.* [47] which combines a virtual dissection method based on model reduction in Section 2.3.2.

2.2.2 Meshfree Methods

Smoothed Particle Hydrodynamics

Smooth particle hydrodynamics (SPH) was first introduced by Gingold and Monaghan [48] for the simulation of astrophysical phenomena. It was later introduced to the computer graphics community by Stam and Fiume [49] to simulate fire and other gaseous phenomena. Over the years, it has been one of the major concepts for fluid simulation because of its flexibility to simulate versatile effects. Since this thesis mainly focuses on solid simulation, we refer the interested readers to a recent overview by Ihmsen *et al.* [5] that covers the applications of SPH in fluid simulation. In the following, we discuss the existing efforts that apply SPH to solid simulation.

Desbrun *et al.* [50] propose an early SPH approach to model the dynamics of nonelastic deformable objects in the context of computer graphics. They propose a modified ideal gas state equation to keep a constant rest density of the material and therefore change the interaction force between pairs of neighboring particles. The material modeled by their method exhibits some internal cohesion, which is appropriate to model some highly deformable objects. They also propose a "spiky kernel" to avoid over-clustering between particles. Since SPH does not have an explicit definition of the surface, they use the mass density function to define an iso-surface for the deformable object.

Solenthaler *et al.* [51] propose an unified framework to model the interaction between solid and fluid using SPH. Their deformable model extends the work of [52] and [53], which incorporates the continuum mechanics into a meshfree model. However, it approximates the Jacobian of the deformable field using SPH, and therefore handles the coarsely sampled and coplanar particle configurations robustly. The concept of a *locally undeformed object* condition is introduced, and each particle stores a distance vector to each of its neighbors instead of its reference position. The simulation demonstrates the phase transition between rigid, elastic and liquid state of the deformable object that employs the *locally undeformed object* condition. By choosing a large gas constant, their method handles the collision between objects well; but an explicit collision handling might be desired to guarantee no penetrations.

Becker *et al.* [6] extend the work of Solenthaler *et al.* [51] by adopting the co-rotational concept. They point out that the deformation gradient in [51] is not rotationally invariant and therefore the method of Solenthaler *et al.* fails to distinguish rotation from shear pressure even it uses the rotationally invariant strain tensor. Their method adopts the co-rotational idea that is used in mesh-based methods such as the work of Müller *et al.* [40] [41]. The rotation of each particle is calculated based on its initial neighborhood, and the elastic force is computed aligned with the reference configuration. By using SPH, their method can also handle the coplanar or collinear configuration of the neighborhood robustly.

Moving Least Squares Approximations

SPH is a versatile and simple approximation framework to model deformable objects. However, SPH has in general poor accuracy. In practice, it lacks zero order consistency, which

means it does not always recover constant functions. Müller *et al.* [52] introduce a meshfree simulation framework which incorporates the moving least squares approximations (MLS) to approximate the spatial derivatives of the displacement field. By using MLS, the approximation of the Jacobian of the deformable field is first order accurate, which guarantees zero elastic forces for rigid body motions. However, it is more computationally expensive because the inversion of the motion matrix needs to be computed for every sample point. Also, the inversion might fail if the motion matrix is singular, which leads to stability problems.

As we mention in Chapter 1, meshfree methods do not naturally have a well-defined boundary of an object. It is not clear how to combine meshfree methods with haptic interaction. Although it is still possible to apply a meshfree method in our scenarios, a mesh-based method is a more natural choice since it satisfies most of our requirements.

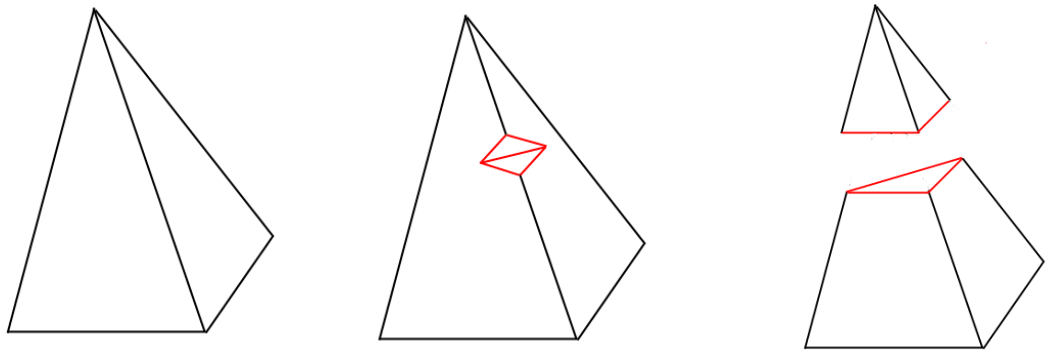
2.3 Dissection Simulation

Dissection is an essential part of many applications, such as surgical simulators and planners. It is worth noting the difference between dissection of deformable objects and that of solid object. In the deformation of a solid object, the distance between all points in the object body remains unchanged. This is also referred as *rigid-body displacement*. As a result, dissection of solid objects can be viewed as a simplified version of the deformable object case, which only involves static geometric and topological changes of the object. On the other hand, in the case of deformable objects, the two separate parts of a dissected object may undergo different deformations during the simulation.

There are a number of research efforts done that are trying to solve the virtual dissection problem of deformable objects. However, due to the complexity of this problem, there is no perfect solution available yet. A physically accurate method is preferred to simulate the dynamic behavior of a dissected object after an arbitrary shaped cut. Also, in an interactive scenario, we require the simulation to be robust enough in order to deal with unpredictable user inputs, such as thin slicing and repeated cuts at the same locations. Last but not the least, in a real-time application, the performance of the simulation is very important; a large delay would definitely hurt the sense of immersion. Most literature on this topic makes different tradeoffs in their approaches to achieve different goals. We focus more on performance over accuracy to achieve real-time simulation. Also, since our system includes haptic interactions with users, stability and robustness become very important requirements.

It is also worth noting the difference between a progress dissection and a non-progressive one. Generally, the progress of cutting an element involves three phases: (1) before cut, (2) partial cut, and (3) complete cut (See Figure. 2.1). In an ideal dissection simulation, the cutting during the simulation should be fully progressive, which means a partial cut would change the topology of the deformable object immediately. As a result, there is possibility that the simulation need to deal with topological changes of the deformable object in every time step, which would have a huge impact on the performance and therefore limit its application in a real-time scenario. As the opposite, a static cut means the splitting surface

must be fully determined before any topological changes are applied to the deformable object. The implementation of static cutting is simple, and there is no performance overhead during cutting in the simulation. However, a static cut can hardly give feedback to the user during the cutting interaction, which definitely hurts the immersion. As a compromise, in our system we adapt the notion of semi-progressive cutting, which is also mentioned in the work of Jeřábková and Kuhlen [17]. The core idea of semi-progressive is to only deal with complete cuts of elements (i.e. an element is cut into two separate parts) and ignore any partial cuts of the elements. It avoids the computational overhead caused by the topological changes of the partial cut elements, and in the case of small elements, it still produces convincing visual dynamic behavior of the dissected object.



(a) Original Tetrahedron (b) Partial cut (c) Complete cut

Figure 2.1: Different Cut Patterns

The choice of the volumetric mesh has direct impact on the performance and the stability for the virtual dissection simulation of a mesh-based method. There are several representations of the volumetric mesh that are popularly used in research. Tetrahedral meshes are used extensively in early works in the field because they give great flexibility in modeling. Combined with linear shape functions, the strain field is piecewise constant over the volumetric domain, which can be precomputed and therefore reduce the online computational work. However, most of the virtual dissection methods based on tetrahedral meshes model the cutting procedure by manipulating the existing tetrahedra Therefore, it is prone to create the ill-shaped elements that hurts the stability of the simulation. There are some recent works trying to avoid this issue by using regular or semi-regular meshes, such as hexahedral meshes. However, unlike tetrahedral meshes, hexahedral meshes do not have a well-defined triangulated surface for rendering or other purposes, which requires additional efforts to construct the surface from the volumetric domain. Unlike mesh-based methods, meshfree methods do not require a simulation grid to model the objects. Instead, the material is only represented by a set of moving particles. The absence of the geometric and topological encoding provides great flexibility to model the changes of objects. There are also efforts to apply meshfree methods to model dissection process of deformable solids.

2.3.1 Mass-Spring Systems

There are many attempts to model dissection using mass-spring systems. Most of them model dissections by removing or refining the connectivity of the particles within the volumetric mesh.

Bielser *et al.* [54] apply a mass-spring system on a tetrahedral mesh. Their dissection method operates tetrahedral decompositions to responsively change the topology and geometry of the network of springs. An intersected tetrahedron is subdivided into a set of 17 smaller tetrahedra, which involves substantial topological changes to the underlying spring network. In order to resolve collision between the virtual scalpel and the deformable object, a local collision detection method to search interior intersected tetrahedra is proposed.

Bielser and Gross [55] extend the work of [54] by only subdividing the edges and faces that are part of the cutting surface. The permitted subdivision of a tetrahedral face is restricted by three cases. Then the cut algorithm contains five different patterns, where there are two patterns of complete dissection and three for partial dissection. They also improve a volume collision detection method which can detect all the intersected tetrahedra against a cutting surface instead of only those that are directly adjacent.

Zhang *et al.* [56] propose a progressive cutting method of a deformable object based on an extended surface mass-spring mesh. In order to model the cuts on a surface mesh, their methods progressively subdivides and re-meshes the mass-spring system. The side topology is generated on both sides of a cut. In partially cut scenarios, a "groove" is also generated at the bottom of a cut in order to connect the side topology on both side of the cut. to generate for partial cuts and side topology for complete cuts. Their method achieves high update rates and produces visually convincing result. However, their method is based only on the surface of the deformable object and not a volumetric representation of the object. For scenarios where objects are inhomogeneous and have different properties between the surface and the interior volumetric domain, it is hard to model accurate material behavior only with a surface model.

Although mass-spring systems are not suitable to our scenarios, there are some general ideas that can be applied to our system. Since the dissection modeling on the mass-spring systems is to modify the spring network, it is necessary to detect the springs that are intersected by the cutting tool. This detection technique can also be applied to our system, and our collision detection method is based on the work of Bielser *et al.* [54] and Bielser and Gross [55].

2.3.2 Finite Element Method

Remeshing Methods

Mor and Kanade [12] present a progressive cutting method based on a linear FEM as a follow up work to [54]. A hybrid method of surface and volume collision detection is used to find the intersected tetrahedra by the sweeping surface of the cutting edge. The intersected elements are then subdivided accordingly and replaced with a set of new elements for each

element. Substantial computational resources are spent on changing the topology of the underlying volumetric mesh. For large cuts, the number of tetrahedra is also increasing rapidly, which might not be desirable for a real-time interactive application. The main issue of this re-meshing method is that it is possible to produce ill-shaped elements (e.g. almost zero volume) which hurts the stability of the simulation.

Cotin *et al.* [57] propose a hybrid method of the quasi-static pre-computed elastic model and the dynamic tensor-mass model to build a real-time surgical simulator. The cutting operation in their model is to remove the tetrahedra touched by the virtual scalpel. However, the surface after element deletions does not conform to the cutting surface created by the cutting tool. It is possible that the deletions might leave holes on the surface, which is not visually satisfying. More importantly, this method is not physically accurate, since simply removing tetrahedra along the cutting path violates the mass preservation law. The loss of volume changes the dynamic behavior of the deformable object as the number of the deleted elements increases.

Nienhuys and van der Stappen [58] propose a virtual dissection method by restricting the cuts to be performed along faces. They introduce a heuristic method to find the faces of the tetrahedral mesh that are near the splitting surface created by the cutting tool. The cutting process on the deformable object is performed on the topological level. Therefore, they use an iterative method to solve the system without explicitly calculating the stiffness matrix, which changes due to the topological changes. The main problem of their method is that the cutting surface is not guaranteed to be a smooth surface, because it is aligned with the faces of the volumetric mesh. Therefore, the resulting surface depends on the geometric feature of the tetrahedral mesh and often becomes jagged in practice.

In order to construct the new surface that conforms with the cuts, Nienhuys and van der Stappen [13] propose a modified method that snaps nodes of the mesh to a trajectory along the cutting path. The deformable object is modeled by the static linear FEM. The cuts are still performed along the faces of the volume; therefore no additional elements are created during the cutting process. Although their method does not perform any subdivision at all, it is still possible that the node snapping generates elements with an ill-conditioned shape. The degenerate elements must be removed after each cut using edge-collapses. Their method also involves the tetrahedra subdivision for the degenerated elements. However, they do not mention how they update the stiffness matrix after the removal and subdivision of the degenerated elements.

Previous remeshing methods are operated on tetrahedral meshes. While tetrahedral meshes offer great flexibility in modeling physical domains, it is possible to generate slivers and ill-shaped elements after dissections. In our system, a haptic device is used as the interface to the user. Since the user has enough freedom to interact with the deformable object, there is no way to prevent him from the dissections that would generate slivers and ill-shaped elements. Therefore, we need a virtual dissection method that is stable even when the user input is unpredictable and uncontrollable.

Jeřábková *et al.* [59] propose an interactive virtual dissection method based on hexahedral meshes. The volumetric domain of the deformable object is modeled by two level hierarchy of hexahedral elements. Each cell on the fine level represents a portion of mate-

rial. Then each finite element is created by an independent connected graph component of the fine level elements on the coarse level. Since it is possible that a cell of the coarse level is only partially filled, the stiffness within a cell is distributed non-uniformly. The simulation is based on the coarse level elements, while the visualization and collision detection are performed on the fine level. The cuts are performed by removing the elements of the fine level and then updating the elements of coarse level. Their system supports haptic rendering for both repulsion and cutting using a nested geometric model for the cutting tool. The GPU-based implementation can produce visually convincing results in a real-time application. However, since their model of dissections is based on element deletion, it is inevitable that the loss of volume and mass becomes noticeable in the scenarios with large cuts.

Seiler *et al.* [60] propose an interactive cutting method based on non-conforming adaptive hexahedral meshes. The hexahedral mesh that models the deformable object is refined up to the resolution defined by the user adaptively as a preprocessing step. The cutting method is non-progressive, and the cutting tool is defined as a static surface. The cuts are performed in the rest configuration of the deformable object with the geometry of the cutting tool is transformed into the material space. The octree representation is refined in order to update the resolution after cuts. A series of node splits are performed to create new DOFs to model the discontinuities

Dick *et al.* [61] present a hexahedral finite element methods that models cuts by adaptively refining the intersected elements. The deformable object is discretized adaptively using an octree grid, where the leaves represent the finite elements. Adjacent elements are connected via links. The elements that are intersected by the cutting tool are recursively subdivided by a regular octree refinement. On the finest level, the links intersected by the cutting surface are disconnected. Therefore, the cuts are essentially along the faces of the hexahedra. The cutting algorithm is incorporated into a geometric multi-grid scheme that is computationally efficient. However, as mentioned by the authors, the unrestricted refinement of the octree might results in creating more elements than would actually be required to solve the system accurately.

Wu *et al.* [62] summarize their efforts in building a highly efficient haptic cutting simulation based on hexehedral meshes. A linked volume representation is used to describe the topology of the objects. The cuts are modeled by disconnecting the links that are intersected by the splitting surface between elements. Their method uses a hexahedral octree grid that starts as a uniform grid and is adaptively refined after cuts. In order to reduce the number of DOFs, the composite finite element method is used to merge topologically connected hexahedral elements into a coarse cell. A collision detection specific to the composite finite element method [63] is employed. A geometric multigrid solver proposed by Dick *et al.* [61] is used to solve the linear system. The haptic rendering uses a damping force model to generate force feedback.

Virtual dissection methods based on hexahedral meshes in a popular trends in recent years. Its (semi-)regular structure ensure an effective mean to model cuts without worrying the generation of slivers and ill-shaped elements. However, in order to compensate the jagged surface of the hexahedral meshes, a separate surface presentation is required. And in

general, tetrahedral meshes is easier to generate for complex geometric features compared to hexahedral meshes. Therefore, our system is based on tetrahedral meshes. But the virtual dissection method based on the hexahedral meshes is a promising direction.

Virtual Node Algorithm

Molino *et al.* [14] propose a virtual node algorithm that models the cut by creating several replicas of the intersected element. Unlike the re-meshing methods, which subdivide the intersected elements into a set of new elements, the virtual node algorithm creates additional virtual nodes and constructs new elements from the virtual nodes and their neighboring nodes. Their method is based on the assumption that each node in the tetrahedral mesh represents the portion of the material around it. Each replica represents a portion of the material from the original element, which depends on the number of real nodes within the replica. The issues of slivers are alleviated because each sliver is nicely shaped. In order to deal with both degenerate and inverted elements, they use the diagonalized finite element method from the work of Irving *et al.* [39]. Their result demonstrates that this method can separate the mesh along arbitrary piecewise linear paths. However, the performance of virtual node algorithm (it takes several minutes per simulation for 1K triangles in [14]) is more appropriate for off-line applications rather than real-time interactive simulation.

Sifakis *et al.* [15] extend the virtual node algorithm into a more general method that can support more cutting patterns including arbitrary cutting of existing cuts and progressive cuts during deformations. The deformable object is modeled by a set of two meshes: a coarse tetrahedral mesh for simulation, and a high resolution triangulated mesh for rendering and cutting. Unlike the work of Molino *et al.* [14] which requires that each replica of an intersected element must have one real node at least, their method allow for replicas that formed by all virtual nodes. Therefore, the virtual dissection is not limited by the resolution of the tetrahedral mesh and supports an arbitrary number of fragments within a tetrahedron. The cuts are introduced in the form of triangulated surfaces. The cutting triangles are resolved into an intersection-free polygonal mesh, which is eventually triangulated for collisions and rendering. The tetrahedral mesh for simulation is updated by the same topological changes as that of the surface mesh. However, it is not clear how the masses of the fragments are updated in their method.

The methods based on virtual node algorithm is flexible to model complex dissections. However, there is no experiments to demonstrate how the integration time changes as the number of elements increase during the simulation. Also, based on their performance, it is more suitable to off-line applications than interactive simulations.

Extended Finite Element Method

Some researchers have worked to use the extended finite element method (XFEM) in computer graphics. The XFEM extends the classical FEM by enriching the solution space of the underlying system with discontinuous functions. The degrees of freedom of intersected elements are doubled in order to model the deformations of both separated parts.

Although additional vertices are appended to the system as the cut proceeds, the original volumetric mesh remains unchanged. The cost of topological changes that appear in most of the methods mentioned above is therefore avoided.

Kaufmann *et al.* [16] adapt the XFEM in their simulation of cutting and fracturing thin shells. They propose harmonic enrichment functions which can handle general cuts within an element, e.g. multiple, intersecting, arbitrary shaped, progressive cuts. The enrichment functions are stored in enrichment textures to visualize fracturing on a finer resolution mesh while simulating on a lower resolution mesh. However, their method focuses only on cutting on two-manifold surface, which is not appropriate to simulating deformable solids. Their method is also designed to be statically solved and it would need to be re-designed for dynamic simulation.

Jeřábková and Kuhlen [17] present a real-time surgical simulation based on XFEM. By using XFEM, the cuts are modeled without topological changes to the tetrahedral mesh. Their method is able to avoid slivers or ill-shaped elements created during a cut, which can substantially improve the stability of the simulation. The cut paths are not required to be aligned with the elements, which is also suitable for real-time interactive simulation because the user inputs are somewhat unpredictable. However, their method does not support multiple cuts within a single element. Although, they give a brief discussion of describing multiple cuts within an element under the general XFEM mathematical framework.

Niroomandi *et al.* [47] propose a combined method of model reduction and the XFEM framework. By using the proper orthogonal decomposition (POD) method, a much smaller system is derived from some initial off-line simulations. They demonstrate that the stiffness matrix of a cornea mesh with more than 8000 vertices and 7000 elements can be reduced to a 15×15 dense matrix, which can be solved efficiently in real-time. The concept of locally super-imposed patches is introduced to couple the globally supported shape functions in reduced-order modeling in the XFEM. However, in order to achieve real-time performance, their method makes several simplifications on the XFEM. Their method is based on the linear FEM, and does not update the stiffness matrix during the simulation. While these simplifications can provide better performance, it leads to higher errors in the result.

Our system is based on the 3D XFEM framework from the work of Jeřábková and Kuhlen [17], which we believe that we show to deliver stability and robustness in a haptic simulation that may involve fairly unpredictable inputs. Using XFEM in a semi-progressive cutting setup means it avoids the large computational overhead of frequently changing the topology during the simulation. And by using tetrahedral meshes that have "small enough" elements, the visual appearance of cutting dynamics is still convincing.

2.3.3 Meshfree Methods

Pauly *et al.* [9] present a meshfree method to animate fracture effects of elastic and plastic material for offline applications. Their method is based on the point-based animation from the work of Müller *et al.* [52]. The discontinuities are modeled by adapting the shape function using the transparency method, which essential lengthens the interaction distance between two nodes that are separated by a crack. The boundary of a deformable object is

defined by a collection of surface samples. As the crack propagates, the resolution of the crack nodes and the surface samples are refined by adding new samples.

Steinemann *et al.* [10] present an efficient method for splitting of deformable objects that combines the explicit mesh-based surface representations with the meshfree simulation framework. Their method generates an explicit surface from the swept surface of the cutting or fracture fronts. The point samples are updated using the explicit swept surface as the crack propagates. A visibility graph that stores proximity information is used to estimate the distance along fully visible paths between two point samples. Their method also use the MLS to approximate the displacement field from the work of Müller *et al.* [52].

Pietroni *et al.* [11] introduce a tessellation algorithm for a growing, deformable surface that is embedded in a regular grid. The tessellation is based on the intersection between the surface and the edges of the grid. The positions of the tessellation vertices are interpolated within the grid, therefore it is possible to model discontinuities within a cell. They propose the extended visibility criterion to cope with the artificial discontinuity from the original visibility criterion. The line of sight between two point samples is modeled as a pair of cones, and the common base of the cones is defined as the visibility disk. The shape function is then augmented by the ratio of the visible region within the visibility disk.

LeBlanc *et al.* extend the work of Solenthaler *et al.* [51] by introducing plasticity to the deformation. In their method, the strain is decomposed into an elastic and a plastic components. In order to allow plasticity and fracture, the reference position and neighborhood of each particle are updated in each integration step. If the distance between a particle pair exceed a user defined limit, both particles remove the other from the respective neighborhoods to model the fracture. They present a CUDA-based SPH engine that employs the computational power of the GPU. Their system is able to simulate about 6000 simulation nodes and 6000 surface nodes at 44 frames per second.

2.4 Haptic Rendering

Haptic rendering is an essential component in an interactive surgical simulation. Studies have shown that tactile feedback can increase the sense of presence in virtual environments [64]. In conjunction with visual feedback, haptic training might be an effective tool for teaching skills that have force-sensitive components [65]. In order to achieve stable and realistic force feedback, a high force update rates is required. This poses performance requirements on the collision detection and the computation of a feedback force.

Because of its importance in realizing an ideal realistic simulation, the research on haptic rendering has been active for decades. A detailed introduction or review is beyond the scope of this thesis, and we refer the interested readers to some existing literature instead. The course notes of Otaduy and Lin [66] give a great introduction to the state of the art in haptic rendering techniques and demonstrate a wide range of applications using haptics. Misra *et al.* [67] review the literature of deformable models that handle the interaction between tool and tissue in the context of computer based surgical simulation. Teschner *et al.* [68] summarize state of the art methods in the area of deformable collision

detection that include bounding volume hierarchies, spatial partitioning, distance fields, and others.

2.4.1 Point Interaction

Most of the research on haptic rendering models the interactions between users and the virtual object with a single contact point. Rendering algorithms that follow this description are referred as 3-DoF haptic rendering algorithm because a single contact point only has three degrees of freedom [69]. The contact model as well as the force rendering model is simplified in 3-DoF haptic rendering, because the torques arising from the interaction do not need to be computed and there is no torques feedback in the haptic response. There is already much existing work on 3-DoF haptic rendering algorithms.

The basic approach for point based haptic rendering is penalty-based rendering. When the haptic interaction point (HIP), which is the endpoint location of the physical haptic interface in the virtual environment, is inside the virtual object, a penetration depth is computed and the haptic force is proportional to the penetration depth. Essentially this is a case of Hooke's law and can be viewed as attaching a spring between the HIP and the surface of the virtual object. However, the main issue of this simple approach is the difficulty to enforce geometric accuracy. First, it is hard to enforce the non-penetration constraint to keep the HIP on or at least close to the surface of the virtual object. If we assign a very large spring constant, then the system might become unstable because of the large stiffness. In addition it could generate haptic forces that have very large magnitude and overpower the haptic device. Secondly, it is very likely that the HIP can go through thin parts of a virtual object without any haptic resistance, even if the virtual object is supposed to be very stiff. In order to overcome the weaknesses of penalty based rendering, there are more sophisticated haptic rendering models.

Zilles and Salisbury [70] propose the god-object algorithm to deal with the non-penetration constraint for rigid polyhedral models. Their method does not try to prevent the HIP from penetrating the virtual object, but they define the god-object to represent the virtual location of the HIP that is constrained on the surface of the virtual object. The location of the god-object is chosen to be the point on the virtual object's surface that minimizes the distance between the HIP and the god-object. The haptic force is then computed using the distance between the HIP and the god-object with Hooke's law.

Similar to [70], Ruspini *et al.* [71] use a virtual proxy to represent the HIP in the virtual environment. But the virtual proxy method extends the original god-object algorithm so that it can model more behaviors such as force shading, friction, surface stiffness and texture by changing the position of the virtual proxy. During the process of updating the position of the virtual proxy, it is possible that the moving path of the virtual proxy intersects with obstacles in the virtual environment. They use the configuration space of the proxy, which consists for spherical tools of the space within one proxy tool radius of the original obstacles, to model the interactions. After all constraint planes are found from the intersection tests, the new position of the virtual proxy is found by using Lagrange

multipliers similar to [70]. They also describe other haptic effects such as force shading and frictions based on the virtual proxy method.

Zhuang and Canny [72] present a haptic simulation system that provides haptic interaction with deformable objects with large deformations. Their haptic rendering method also use the notion of virtual proxy to deal with the HIP penetrating the virtual object. But the collision between the HIP and the virtual object is resolved by enforcing a non-penetrating constraint that the relative velocity between the HIP and the contact surface is zero. And instead of simply minimizing the distance between the HIP and the location of virtual proxy as [70], the position of the virtual proxy is influenced by the force from the collision.

2.4.2 Haptic Cutting

Haptic rendering of cutting has been a challenging research topic because of the complexity of this problem. In addition, as we discuss above, dissection simulation on deformable objects is still an active research topic. The existing methods might either lack performance to be used in real-time applications or suffer from instability issues. The requirement of update rate in haptic applications (approximately 1000Hz for realistic feedback) makes it even harder to solve. An ideal haptic rendering method for cutting should be physically accurate while not computationally costly because of the high frequency of force computation. Although it sounds very difficult to solve perfectly, there is already some pioneer work done in this field.

Mahvash and Hayward [73] model the cutting on deformable objects as an interchanging process between three forms of energy: the potential energy of the deformable object, the work done by the cutting tool, and the irreversible energy spent in creating a crack. As a result, three types of interaction appear in their method: deformation, progressive cutting, and fracture. The force is computed based on the cutting tool displacement, the roughness of the material, and the geometry of the surface. By tuning the material parameters, their method predicts plausible force feedback that shows the interchange phase of energy in comparison experiments on potato samples. However, their method fails in their experiments with a bovine liver, which shows that their energy based model is incomplete for the force response of deformable object cutting.

Okamura *et al.* [74] present haptic scissors which is a two degrees of freedom device to creating the sensation of cutting in virtual environments. They propose an analytical model that computes the haptic force response based on friction, material properties, and user motion. In their study, they show that users cannot differentiate the generated force feedback between the analytical model and an empirical model using real tissue data. Mahvash *et al.* [75] extend the haptic scissors with an energy based model similar to their previous work [73]. In their new analytical model, two types of interactions are considered: deformation and sharp cutting. During the deformation phase, the haptic force is computed according to the location of the crack edge and the curve of the blades. During the sharp cutting phase, the force is computed based on the following information: the location of

the crack edge, the openning angle of the blades, and the material toughness. Their model shows accurate predictions of the average haptic response force.

Chapter 3

Background

In this chapter we focus on modeling the deformations of elastic bodies in a 3-dimensional space. The deformations of an objects can be analogous to similar concepts from mass-spring systems. However, a volumetric body can deform in more complex ways that requires more expressive mathematical framework. Our discussion will focus on basic concepts of continuum mechanics and finite element discretization. Techniques that deals with large displacement and describes discontinuities caused by cuts in the framework of linear finite elements method are also covered.

The full coverage of this topic is beyond the scope of this thesis. We refer the interested readers to some other literature for more detailed information. Witkin and Baraff [76] give a great introduction to continuum mechanics and dynamics. Müller *et al.* [77] discuss the existing physically-based simulation methods used in real-time applications. Sifakis and Barbic [78] provide a detailed introduction to the FEM and its mathematical derivations.

3.1 Continuum Mechanics

In this section, we will provide a brief introduction to the mechanics of deformable objects using linear elasticity. The study of the kinematics and the mechanical behavior of a continuum material is known as continuum mechanics. It provides a mathematical framework to model the dynamic behavior of materials under certain conditions.

Generally, the behavior of materials can be characterized by a set of partial differential equations (PDE) that describe the states of displacements, strain and stress of each point within the material body. It consist of:

- Static equilibrium relating the external force and stress;
- Kinematic compatibility condition relating the strain and displacement;
- Constitutive equation relating the stress and strain; and
- Boundary conditions which includes displacement and force constraints.

As shown in Figure 3.1, these relationships allow us the model the deformation of a deformable object under certain boundary constraints. In the following, we mainly focus on the discussion of linear elastic material which uses Hooke's law as the constitutive equation. Also, we restrict our discussion to *homogeneous isotropic* materials, which are much simpler to model and can be solved efficiently in a real-time application.

To a certain degree, the following discussion is analogous to similar concepts from a 1-D mass-spring system. However, in the case of volumetric mesh the deformation is much more complex than an elongation in the case of a spring. As a result, the formula need to be extended to express the mathematical description of deformation in higher dimensions. We give a brief introduction of continuum mechanics in the case of linear elastic material to help our later discussion of FEM.

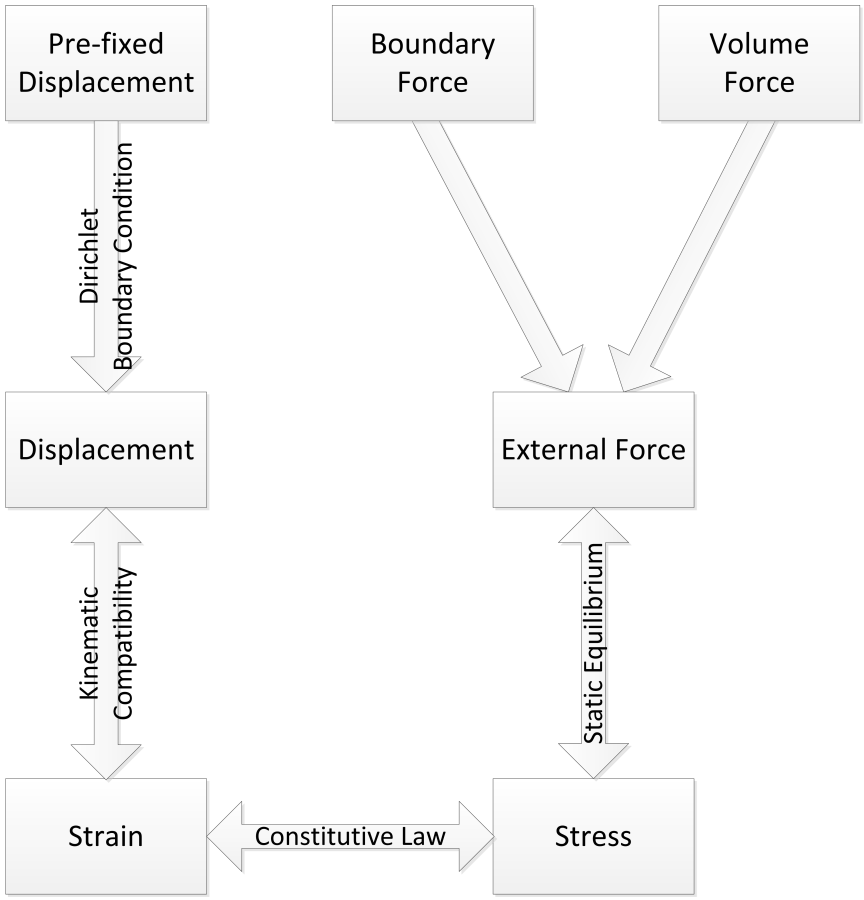


Figure 3.1: Relationships between the Physical Properties of a Material

3.1.1 Deformation

A deformable object is typically defined by its undeformed shape, which is often referred as the *reference configuration*. Assume an undeformed deformable object is placed in a coordinate system and occupies a certain volumetric domain Ω, and $\mathbf{p}^0 \in \Omega$ denotes

the location of a material point in the reference configuration. In a continuous body, a deformation may be caused by an external force load, a global force field or a change of material behavior. Then the deformation can be described as a transformation of the body from the *reference configuration* to the current configuration. In the current configuration, every material point $\mathbf{p}^0$ is mapped to another spatial locations, $\mathbf{p} \in R^3$. The relation between $\mathbf{p}^0$ and $\mathbf{p}$ can be expressed by a *deformation function* $\phi : R^3 \rightarrow R^3$. If we denote the displacement field of the material points in the deformable object as $\mathbf{u}$, then $\mathbf{p} = \phi(\mathbf{p}^0) = \mathbf{p}^0 + \mathbf{u}$.

3.1.2 Strain

Strain is a quantitative description for the severity of the deformation. It is an analogous concept to the ratio of elongation with respect to the original length in a 1-D spring case. However, in the 3-D case, strain ε becomes a 3×3 symmetric matrix (see Equation 3.1) and is defined as the ratio of deformation to the reference configuration. As a result, strains are derived from the deformation gradient $\mathbf{F} = \nabla\phi = \mathbf{I} + \nabla\mathbf{u}$ which only depends on the change of shape. Note that there are two important assumptions under the statement above: (1) the strain is fully determined by the current deformation and not by the prior deformation history; And (2) the total work done by the internal elastic forces during the deformation only depends on the reference and current configurations.

$$\varepsilon = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix} \tag{3.1}$$

A useful measure of strain is the *Green's strain tensor* $\varepsilon_{\mathbf{G}}$ which is defined as

$$\begin{aligned} \varepsilon_{\mathbf{G}} &= \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) \\ &= \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T + \nabla\mathbf{u}^T\nabla\mathbf{u}) \end{aligned} \tag{3.2}$$

The Green's strain tensor has many good properties we would look for in modeling material dynamics:

- When $\mathbf{u} = \mathbf{0}$, $\varepsilon_{\mathbf{G}} = \mathbf{0}$, which means it has the minimum energy in its reference configuration.
- When $\phi(\mathbf{p}^0) = \mathbf{R}\mathbf{p}^0 + \mathbf{t}$, where $\mathbf{R}$ is a rotation matrix and $\mathbf{t}$ is a translational vector, $\varepsilon_{\mathbf{G}} = 0$, which means it is rotationally invariant.

However, by using the Green's strain tensor, the relationship of the internal force and displacement becomes nonlinear. This would increases the complexity of the constitutive models that are based on the Green's strain tensor. It is impossible to solve such a non-linear PDE system using an implicit integrator scheme, and using the Green's strain tensor requires the use of explicit integrator methods which are not guaranteed to be stable.

By assuming the deformations are small ($|\mathbf{u}| << 1$), a *linear approximation* $\varepsilon_{\mathbf{C}}$ of equation 3.2 can be constructed by forming a Taylor expansion around the reference configuration $\mathbf{F} = \mathbf{I}$ as

$$\begin{aligned} \varepsilon_{\mathbf{G}} &\approx \mathbf{E}(\mathbf{I}) + \frac{\partial \mathbf{E}}{\partial \mathbf{F}}|_{\mathbf{F}=\mathbf{I}} : (\mathbf{F} - \mathbf{I}) \\ &= \varepsilon_C = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \end{aligned} \tag{3.3}$$

ε_C is the *infinitesimal strain tensor* or Cauchy linear strain tensor. By omitting the quadratic term $\nabla \mathbf{u}^T \nabla \mathbf{u}$ in Equation 3.2, the Cauchy linear strain tensor is linear to the displacement. As a result, the relationship of the internal force and displacement becomes linear as well, which makes using an implicit integration scheme possible. However, this is at the cost of only being valid for small strains. The Cauchy strain tensor causes significant artifacts in a scenario with a large rotational deformation.

3.1.3 Stress

In continuum mechanics, stress σ describes the internal force distribution that exerts on a point's neighboring in a deformed configuration. A deformation of a solid object would generate an internal stress for each mass point in the body, which is analogous to the reaction force of an elongated or compressed spring. Just as strain in the 3-D case, stress can be presented as a symmetric 3×3 matrix or tensor for each point in the material body (see Equation 3.4). The diagonal entries are called the orthogonal normal stresses, and the other entries the shear stresses.

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix} \tag{3.4}$$

The stress at a material point in the body depends on the direction of measurement. Let $\mathbf{n}$ be the unit normal vector in the direction of measurement, the stress vector across an imaginary surface is defined as

$$\frac{d\mathbf{f}}{dA} = \sigma \cdot \mathbf{n} \tag{3.5}$$

3.1.4 Constitutive Equation

The constitutive equation relates the stress and strain of a material. It determines the actual material behavior because it defines the response of the deformable object under a given stimuli. Different constitutive equation are used to describe different material behaviors, such as elasticity, viscosity, and plasticity. In the field of computer graphics,

Hooke's Law is very popular to model linear elastic material. It relates the stress and strain with a linear function as

$$\sigma = \mathbf{E} \cdot \varepsilon_{\mathbf{C}} \tag{3.6}$$

where $\mathbf{E}$ is a 6×6 matrix or stiffness tensor. For isotropic materials, $\mathbf{E}$ is a constant matrix that only depends on two material properties:

- Young's modulus μ that describes the elastic stiffness and
- Poisson's ratio ν that describes the compressiveness of the material.

As a result, $\mathbf{E}$ is a constant matrix where

$$\mathbf{E} = \frac{\mu}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2\nu \end{bmatrix} \tag{3.7}$$

This is an important observation to its application of real-time simulation because $\mathbf{E}$ can be pre-computed before the simulation and therefore less computation work need to be done during the simulation.

3.1.5 Linear Elasticity

With the definitions and assumptions above, we are able to model the dynamic behavior using Newton's second law of motion. Assume an infinitesimal volumetric element be at the location $\mathbf{x}$ of the deformable object, then the acceleration of this material point results from the sum of the internal force and the external point at its location as

$$m\ddot{\mathbf{p}}(\mathbf{x}) = \mathbf{f}_{int}(\mathbf{x}) + \mathbf{f}_{ext}(\mathbf{x}) \tag{3.8}$$

where $\mathbf{f}_{int}$ is due to the deformation and can be calculated by integrating the stress over the element surface. Based on the two linear assumptions we discuss above: (1) *geometric linearity* that relates the deformation and strain linearly, and (2) *material linearity* that relates the stress and strain linearly, Equation 3.8 becomes a linear PDE. Materials that are modeled by Equation 3.8 are referred to as *linear elastic material*. It is a simplification of the more general non-linear elasticity equation given the above assumptions. It has a simpler computation process at the cost of being only valid in small deformation scenarios. However, because a PDE system based on linear elasticity is much easier to solve compared to a non-linear one, it is very popular in real-time applications which require responsiveness even at the cost of physical accuracy.

3.2 Finite Element Method

From the discussion above, we have a complete mathematical framework for modeling deformations for a continuum. However, it is impossible to solve analytically a set of PDEs for a deformable object with all but the simplest shape. In a computer-based simulation, the deformable object need to be sampled at a finite number of points first and then solve the PDEs using numerical methods.

The finite element method (FEM) is one of the most popular methods in computational science to solve such a PDE system defined for a certain domain and with some boundary conditions. In order to adapt this method in the simulations of deformable objects, the object is viewed as a continuous connected volume and discretized into a finite number of sub-domains, which are often referred as *elements*.

There are different volumetric representation existing, and in our project, we use tetrahedral meshes which is very popular in the field of computer graphics (see Figure 3.2 rendered by TetView from [79]). In this case, the physical domain of the object is subdivided into a set of tetrahedra by grouping 4 material sample points in an element. Each element is a sub-domain of the original domain and is represented by its own Equation 3.8 locally without considering neighboring elements. Then all sets of equations are combined into a global system for the final calculation, where the connectivity of elements takes effect. To a certain extent, it is an analogous concept to the mass-spring systems where pairs of mass points are connected by one dimensional springs. However, in the case of FEM, a set of four mass points is connected to others, and the relationship between each pair of neighboring elements can be viewed as a high-dimensional springs. Compared to some simpler deformable models, such as mass-spring systems, the FEM is more sophisticated and computationally more expensive. However, the FEM is more physically accurate; and the behavior of deformations can be specified with several material parameters instead of tuning a large number of spring constants. For the simulations that requires more physical accuracy, such as surgical simulation and medical training, the FEM is a preferable solution method for the underlying deformable model.

The continuous deformation field can be reconstructed from the discrete samples with an interpolation function Φ. Let a deformable object in the reference configuration be defined by a set of N sampled material points $\mathbf{p}_1^0, \mathbf{p}_2^0, \cdots, \mathbf{p}_N^0$. The corresponding deformed material points are $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N$. The relation between $\mathbf{p}^0$ and $\mathbf{p}$ can be expressed by a *deformation function* $\phi : R^3 \rightarrow R^3$. If we denotes the displacement field of the material points in the deformable object as $\mathbf{u}$, then $\mathbf{p} = \phi(\mathbf{p}^0) = \mathbf{p}^0 + \mathbf{u}$. Within an element of the tetrahedral mesh, the displacement of a point $\mathbf{x}$ is interpolated as

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{4} \Phi_i \mathbf{u}_i \tag{3.9}$$

The choice of the interpolation function is closely related to the volumetric mesh representation of the object. Since we focus on tetrahedral meshes, the choice of interpolation scheme is naturally barycentric interpolation, though it is still possible to use more sophisticated interpolation function.

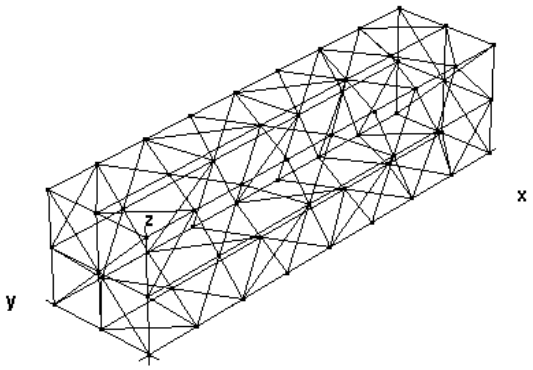


Figure 3.2: Sample Tetrahedral Mesh of a Beam

3.2.1 Linear FEM

Using FEM based on the linear elasticity theory employs the two linear assumption we discuss in previous Section 3.1.5:

- *geometric linearity* states that strain is a linear function of the deformation, which gives

$$\begin{aligned} \varepsilon_e &= \frac{1}{2}(\nabla \mathbf{u}_e + \nabla \mathbf{u}_e^T) \\ &= \mathbf{B}_e \cdot \mathbf{u}_e \end{aligned} \tag{3.10}$$

  where $\mathbf{B}_e$ is a 6×12 constant matrix, and $\mathbf{u}_e$ is the collection of the displacement vectors for a tetrahedron.

- *material linearity* states the stress is a linear function of the strain, which gives

$$\sigma_e = \mathbf{E}_e \cdot \varepsilon_e \tag{3.11}$$

  where $\mathbf{E}_e$ is defined in Equation 3.7 and is also a constant 6×6 matrix that only depends on Young's modulus μ and Poisson's ratio ν.

As a result the relationship between the elastic force and the displacement of an element can be expressed as

$$\mathbf{f}_e = \mathbf{K}_e \cdot \mathbf{u}_e \tag{3.12}$$

where $\mathbf{K}_e = V_e \mathbf{B}_e^T \mathbf{E}_e \mathbf{B}_e \in R^{12\times 12}$ and V_e is the volume of the element. It is worth noting that $\mathbf{K}_e$ is a constant matrix as well, which can be pre-computed before the simulation. This means a great portion of computation can be avoided in the simulation, which is very helpful for real-time applications.

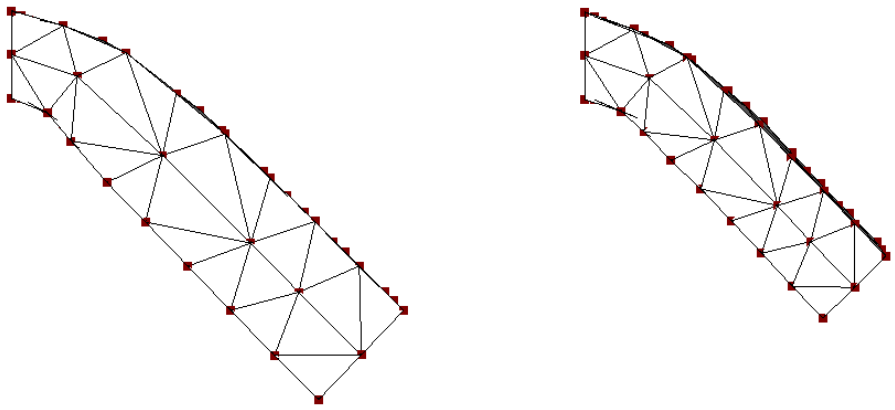
3.2.2 Co-rotational Linear FEM

Although linear FEM demonstrates good efficiency, the fact that it is only valid in small deformation scenarios makes it inappropriate for interactive applications where large deformations can often happen. As we discuss in Section 3.1.2, by using the Cauchy strain tensor, the linear FEM is not rotationally invariant as well. When the deformation of the object contains a rotational component, the strain is not zero and therefore results in artifact internal force. As in Figure 3.3a, it generates an visual artifact that bloats the volume of the object.

Müller *et al.* [41] propose a co-rotational method to overcome the weakness of the linear FEM in a large deformation scenario (see Figure 3.4). The rotational component of the deformation is extracted for each element, and then the deformation is rotated back to the reference configuration. The deformation forces $\mathbf{f}_e$ are computed aligned with the reference configuration as

$$\mathbf{f}_e = \mathbf{R}_e \mathbf{K}_e (\mathbf{R}_e^{-1} \mathbf{p} - \mathbf{p}^0) \tag{3.13}$$

where $\mathbf{R}_e$ is a 12×12 rotation matrix. The exaction of the notational component is the only additional step that different from the original linear FEM, and the efficiency and robustness of this step has great impact on the simulation. There are different options of methods that estimate this rotational matrix. In our project, we follow the method in [41] which calculates the transformation matrix from the reference configuration to the current configuration first, and then use a polar decomposition to extract the rotational component from the transformation matrix. By avoiding the ghost forces caused by rotations, the visual artifact is eliminated (see Figure 3.3b) while still preserving the computational efficiency of the linear FEM.



(a) Linear FEM (b) Co-rotational Linear FEM

Figure 3.3: A Deformable Beam Based on Different Methods

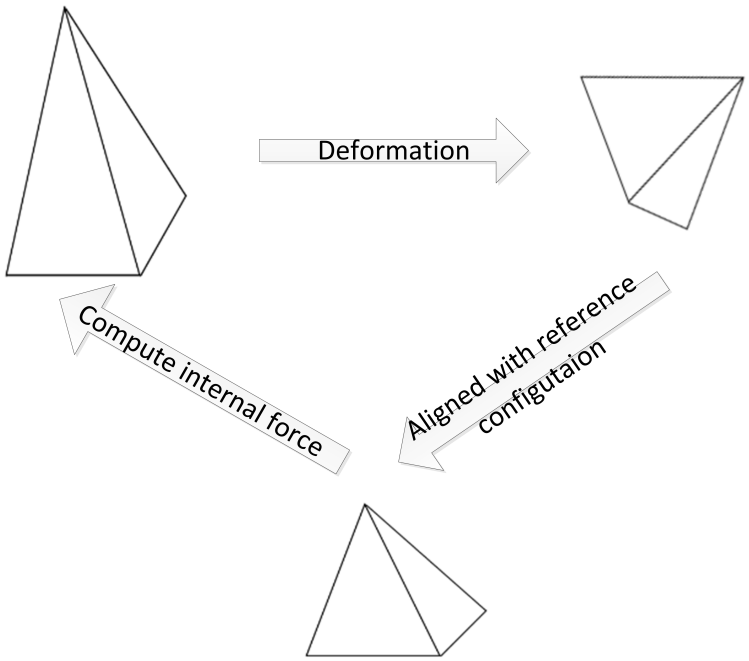
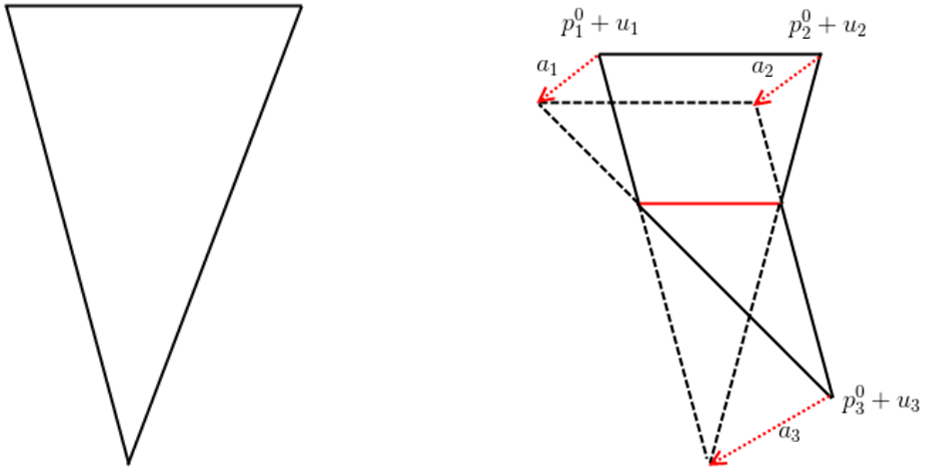


Figure 3.4: The Co-rotational FEM

3.3 Modeling Cuts Using XFEM

In order to build a dissection simulation, we still need a mathematical model for cutting. The extended finite element methods (XFEM) is a method that can effectively model discontinuities within an FEM mesh. It was first used in structure mechanics analysis, and recently introduced to the field of real-time graphics simulations by Kaufmann *et al.* [16], and Jeřábková and Kuhlen [17]. Our project is based on the XFEM framework from the work of Jeřábková and Kuhlen [17] which deal with virtual cuts within a 3-D tetrahedral mesh. We give a brief introduction of how XFEM models cuts and how it can be adapted into the framework of co-rotational FEM.

Figure 3.5 gives a 2-D visual explanation of the XFEM concept. If an element (a triangle in this 2-D case) is cut, then the deformations of the two remaining parts should be independent to each other. Therefore, the degrees of freedom (DOFs) are doubled for this element by adding $\mathbf{a}_e = [\mathbf{a}_1\ \mathbf{a}_2\ \mathbf{a}_3\ \mathbf{a}_4]$ to the displacement field. Essentially, the enriched element becomes two separate elements that undergoes different displacements, and the original vertices of this element belongs to either one of them which depends on which side of the cut they are on. The additional DOFs are used to describe the displacements of the added vertices for these two separate elements. While $\mathbf{u}_e = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3\ \mathbf{u}_4]$ still represents the displacement of the original vertices within this element, the added nodal DOF $\mathbf{a}_i$ represents the offset of the vertex i which "snaps" it to the location within the new element.



(a) Original triangle (b) Enriched Triangle after a Cut

Figure 3.5: A 2-D example of the XFEM

The displacement of a point $\mathbf{x}$ within an enriched element is now dependent on which side of the cut it is on. A global discontinuous enrichment function Ψ is used to define the cut. There are many options of discontinuous functions since it is only required to be discontinuous over the cut domain. However, as discussed in [17], a discontinuous function that satisfies the *Kronecker delta property* is very helpful to simplify the implementation because the nodal displacements of vertices are stored directly in $\mathbf{u}_e$ no matter if they are enriched or not, and $\mathbf{a}_e$ is only needed to determine the displacement within an enriched element. By defining the *sign function* H as

$$H(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is above the crack} \\ -1 & \text{if } \mathbf{x} \text{ is below the crack} \end{cases} \tag{3.14}$$

The *shifted enrichment function* Ψ_i becomes the global discontinuous function:

$$\Psi_i(\mathbf{x}) = \frac{H(\mathbf{x}) - H_i}{2} \tag{3.15}$$

where H_i is the value of $H(\mathbf{x})$ at the i-th vertex. Jeřábková and Kuhlen [17] show that the shifted enrichment function preserves the enrichment contributions of $\mathbf{a}_e$ within an enriched element while eliminating it at the element's border and beyond, which greatly simplifies the implementation. As a result, the displacement $\mathbf{u}^X$ of a point $\mathbf{x}$ within an element in the XFEM becomes:

$$\mathbf{u}^X(\mathbf{x}) = \sum_{i=1}^{4} \Phi_i \mathbf{u}_i + \sum_{j=1}^{4} \Psi_j \Phi_j \mathbf{a}_j \tag{3.16}$$

where we assume that the shape function of enriched DOFs is the same as that of the original vertices.

Because of the introduction of additional DOFs, Equation 3.12 need to be expanded accordingly in XFEM. For an enriched element, the additional offsets $\mathbf{a}$ and the corresponding internal forces $\mathbf{f}^a$ are appended to the original vectors as $\mathbf{u}_e^X$ and $\mathbf{f}_e^X$

$$\mathbf{u}_e^X = [\mathbf{u}_1 \ \cdots \ \mathbf{u}_4 \ \mathbf{a}_1 \ \cdots \ \mathbf{a}_4] \tag{3.17}$$
$$\mathbf{f}_e^X = [\mathbf{f}_1 \ \cdots \ \mathbf{f}_4 \ \mathbf{f}_1^a \ \cdots \ \mathbf{f}_4^a] \tag{3.18}$$

In the linear FEM case, the stiffness matrix $\mathbf{K}_e^X$ is expanded accordingly and then has the following structure [17]:

$$\mathbf{K}_e^X = \begin{bmatrix} \mathbf{K}^{uu} & \mathbf{K}^{ua} \\ \mathbf{K}^{au} & \mathbf{K}^{aa} \end{bmatrix} \tag{3.19}$$

where

$$\mathbf{K}_{ij}^{uu} = \mathbf{K}_{ij} \tag{3.20}$$
$$\mathbf{K}_{ij}^{ua} = (\frac{V_a}{V}\Psi_{aj} + \frac{V_b}{V}\Psi_{bj})\mathbf{K}_{ij} = \begin{cases} -\frac{V_b}{V}\mathbf{K}_{ij} & \text{if } H_j = +1 \\ \frac{V_a}{V}\mathbf{K}_{ij} & \text{if } H_j = -1 \end{cases} \tag{3.21}$$
$$\mathbf{K}_{ij}^{au} = (\frac{V_a}{V}\Psi_{ai} + \frac{V_b}{V}\Psi_{bi})\mathbf{K}_{ij} = \begin{cases} -\frac{V_b}{V}\mathbf{K}_{ij} & \text{if } H_i = +1 \\ \frac{V_a}{V}\mathbf{K}_{ij} & \text{if } H_i = -1 \end{cases} \tag{3.22}$$
$$\mathbf{K}_{ij}^{aa} = (\frac{V_a}{V}\Psi_{ai}\Psi_{aj} + \frac{V_b}{V}\Psi_{bi}\Psi_{bj})\mathbf{K}_{ij} = \begin{cases} \frac{V_b}{V}\mathbf{K}_{ij} & \text{if } H_i = H_j = +1 \\ \frac{V_a}{V}\mathbf{K}_{ij} & \text{if } H_i = H_j = -1 \\ 0 & \text{if } H_i \neq H_j \end{cases} \tag{3.23}$$

In the linear FEM case, it is straight forward to compute the internal force by $\mathbf{f}_e^X = \mathbf{K}_e^X\mathbf{u}_e^X$. The deformation forces of the original DOFs and the added DOFs become

$$\mathbf{f}_i = \sum_{j=1}^{4}\mathbf{K}_{ij}\mathbf{u}_j + (\frac{V_a}{V}\Psi_{aj} + \frac{V_b}{V}\Psi_{bj})\mathbf{K}_{ij}\mathbf{a}_j \tag{3.24}$$
$$\mathbf{f}_i^a = \sum_{j=1}^{4}(\frac{V_a}{V}\Psi_{ai} + \frac{V_b}{V}\Psi_{bi})\mathbf{K}_{ij}\mathbf{u}_j + (\frac{V_a}{V}\Psi_{ai}\Psi_{aj} + \frac{V_b}{V}\Psi_{bi}\Psi_{bj})\mathbf{K}_{ij}\mathbf{a}_j \tag{3.25}$$

If we define

$$\mathbf{p}_{aj} = \mathbf{p}_j^0 + \mathbf{u}_j + \Psi_{aj}\mathbf{a}_j \tag{3.26}$$
$$\mathbf{p}_{bj} = \mathbf{p}_j^0 + \mathbf{u}_j + \Psi_{bj}\mathbf{a}_j \tag{3.27}$$

then the deformation forces can be rewritten as

$$\mathbf{f}_i = \frac{V_a}{V}\sum_{j=1}^{4}\mathbf{K}_{ij}(\mathbf{p}_{aj} - \mathbf{p}_j^0) + \frac{V_b}{V}\sum_{j=1}^{4}\mathbf{K}_{ij}(\mathbf{p}_{bj} - \mathbf{p}_j^0) \tag{3.28}$$
$$\mathbf{f}_i^a = \frac{V_a}{V}\Psi_{ai}\sum_{j=1}^{4}\mathbf{K}_{ij}(\mathbf{p}_{aj} - \mathbf{p}_j^0) + \frac{V_b}{V}\Psi_{bi}\sum_{j=1}^{4}\mathbf{K}_{ij}(\mathbf{p}_{bj} - \mathbf{p}_j^0) \tag{3.29}$$

In the case of co-rotational FEM, we also need to consider the rotation for both parts of the enriched element after dissection (see Figure 3.6). Because of the fact that after dissection, the two separate parts undergo different deformations independently, the rotation matrices need to be extracted with the contributions of the added DOFs. Assume $\mathbf{R}_a$ be the rotational matrix of the part above the cut plane, and $\mathbf{R}_b$ be the rotational matrix of the other part, the internal force is computed as

$$\mathbf{f}_e^X = \mathbf{R}_a \mathbf{K}_e^X (\mathbf{R}_a^T \mathbf{p}^X - \mathbf{p}^{0X}) + \mathbf{R}_b \mathbf{K}_e^X (\mathbf{R}_b^T \mathbf{p}^X - \mathbf{p}^{0X}) \tag{3.30}$$

where $\mathbf{p}^X = \mathbf{p}^{0X} + \mathbf{u}^X$. Similar to Equation 3.28 and 3.29, the deformation forces of the original DOFs and the added DOFs can be rewritten as

$$\mathbf{f}_i = \frac{V_a}{V} \mathbf{R}_a \sum_{j=1}^{4} \mathbf{K}_{ij} (\mathbf{R}_a^T \mathbf{p}_{aj} - \mathbf{p}_j^0) + \frac{V_b}{V} \mathbf{R}_b \sum_{j=1}^{4} \mathbf{K}_{ij} (\mathbf{R}_b^T \mathbf{p}_{bj} - \mathbf{p}_j^0) \tag{3.31}$$

$$\mathbf{f}_i^a = \frac{V_a}{V} \mathbf{R}_a \Psi_{ai} \sum_{j=1}^{4} \mathbf{K}_{ij} (\mathbf{R}_a^T \mathbf{p}_{aj} - \mathbf{p}_j^0) + \frac{V_b}{V} \mathbf{R}_b \Psi_{bi} \sum_{j=1}^{4} \mathbf{K}_{ij} (\mathbf{R}_b^T \mathbf{p}_{bj} - \mathbf{p}_j^0) \tag{3.32}$$

It is worth noting that we only allow a single cut within an element. However, as discussed in [17], it is possible to extend the XFEM framework to adapt the scenarios with multiple cuts.

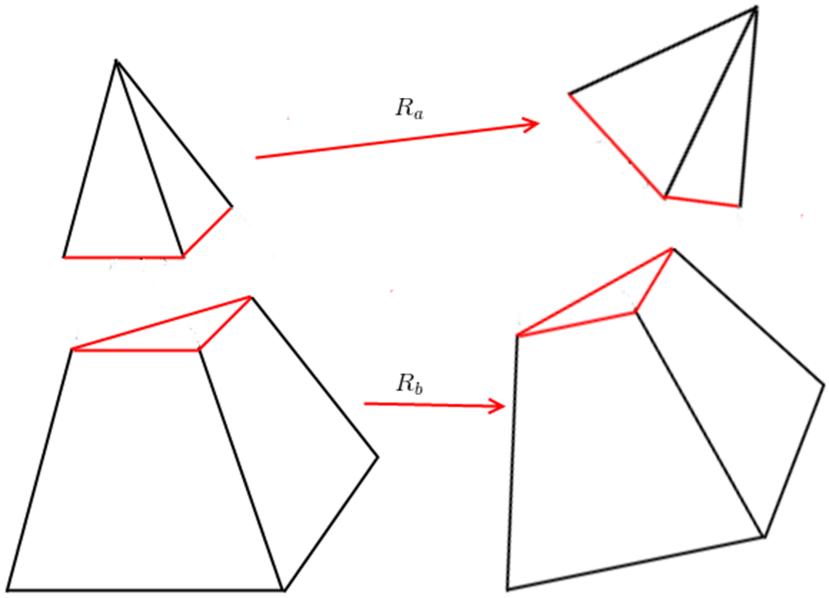


Figure 3.6: Two Components May Undergo Different Rotations

Chapter 4

Simulation System

In this chapter, we provide a high-level overview of the design of our dissection system. Generally, an ideal real-time dissection system should be able to provide the following features [33]:

- efficient simulation of the deformable object's dynamics even with cuts incorporated;
- robust collision detection and topological changes over the deformable object; and
- stable and realistic collision handling via haptic force feedback.

Physically-based collision handling in dissection simulation is still an open research problem, and therefore we will not address the haptic force rendering in our system. Instead, we focus on the design to achieve efficient simulation of the dynamics of the deformable object in a cutting simulation using XFEM. Also, we propose a robust collision detection method to deal with the cutting interaction between the cutting tool and the deformable object. Last but not least, we incorporate point contact into our system to provide another way of interaction other than cutting, and we discuss a local haptic force computation in this scenario which is computationally inexpensive.

4.1 Overview

There are two main tasks of our dissection system: (1) smooth animation of the deformation dynamics, and (2) stable response to the interaction between the deformable object and the haptic tool. And we need to realize that there are different implications of performance for these two tasks. On the one hand, the dynamic simulation should be run at the rate of 60Hz to 120Hz so that it is able to provide enough frames for smooth animation. On the other hand, the haptic thread is usually running at the rate of 500Hz to 1000Hz so that it is able to provide realistic haptic force feedback. As a consequence, the design of our system decouples the deformable dynamics from the haptics. As shown by the high-level system overview (see Figure 1.3 in Chapter 1), we have a dynamic thread to simulate the

object deformation and a haptic thread to deal with the interaction with the haptic device. This design enables that each thread runs with the appropriate update rate.

However, it is worth noting that these threads are not independent of each other. On the one hand, the collision detection and potential collision handling requires information such as the displacement field and the material properties of the deformable object. On the other hand, the dynamic behavior of the deformable object is (partly) driven by the interaction force provided by the haptic thread. The information flow in our system (Figure 4.1) forms a closed loop between the dynamic component and the haptic component: the deformable object undergoes deformation based on the motion induced by the haptic tool; and the haptic force feedback is computed based on the current deformation state of the deformable object and the relative position between the virtual tool and the deformable object.

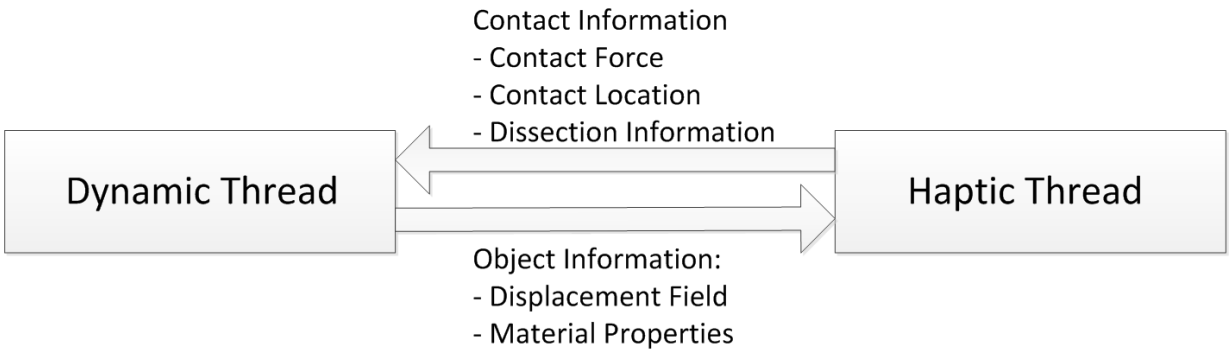


Figure 4.1: Information Flow

In the following sections, we provide a detailed explanation of the components in our dissection system. We discuss the dynamic simulation using XFEM in a cutting scenario in Section 4.2. In Section 4.3, we discuss the collision detection method for point contact and dissection in our system. The haptic rendering of point contacts is covered in Section 4.4. Finally, we show the design of our implementation which leverages multiple open-source libraries in Section 5.1.

4.2 Dynamic Simulation

In our previous discussion in Chapter 3, we model the relationship between the internal deformation force and the displacement of each element in the tetrahedral mesh. In order to solve the deformation of the entire object, the equations of all elements need to be combined into a global system which describes the dynamics of the whole object. Let $\mathbf{a}$, $\mathbf{v}$, and $\mathbf{u}$ be the concatenated acceleration, velocity and displacement fields, respectively, for all the vertices within the tetrahedral mesh. In a real-time interactive simulation, the differential equation for the deformable object is of the form

$$\mathbf{Ma} + \mathbf{Dv} + \mathbf{f}_{int}(\mathbf{u}) = \mathbf{f}_{ext} \tag{4.1}$$

where $\mathbf{M}$ and $\mathbf{D}$ are respectively the global mass and damping matrix.

4.2.1 Tangent Stiffness Matrix

The deformation forces $\mathbf{f}_{int}(\mathbf{u})$ can be computed element-wise by Equation 3.13 for non-enriched elements and by Equation 3.29 for enriched elements. However, in the case of the co-rotational FEM, $\mathbf{f}_{int}(\mathbf{u})$ is a non-linear function of $\mathbf{u}_t$. For implicit integration, we need the *tangent stiffness matrix* $\mathbf{K} = \partial \mathbf{f}_{int}/\partial \mathbf{u}$. Traditionally, as in the co-rotational linear FEM [41], we compute the tangent stiffness matrix of each element as $\mathbf{K}_e = \mathbf{R}_e \mathbf{K}_{e0} \mathbf{R}_e^{-1}$, where $\mathbf{K}_{e0}$ is the linear stiffness matrix. And we assume a linear relationship between the deformation force and the displacement by $\mathbf{f}_{int}(\mathbf{u}) = \mathbf{K}_e \mathbf{u}_e$. In the XFEM case, since an enriched element has two independent components undergoing different deformations, the enriched tangent stiffness matrix is approximated as

$$\mathbf{K}_e = \begin{bmatrix} \mathbf{K}^{uu} & \mathbf{K}^{ua} \\ \mathbf{K}^{au} & \mathbf{K}^{aa} \end{bmatrix} \tag{4.2}$$

$$\mathbf{K}_{ij}^{uu} = \frac{V_a}{V} \mathbf{R}_a \mathbf{K}_{ij} \mathbf{R}_a^T + \frac{V_b}{V} \mathbf{R}_b \mathbf{K}_{ij} \mathbf{R}_b^T \tag{4.3}$$

$$\mathbf{K}_{ij}^{ua} = \frac{V_a}{V} \mathbf{R}_a \Psi_{aj} \mathbf{K}_{ij} \mathbf{R}_a^T + \frac{V_b}{V} \mathbf{R}_b \Psi_{bj} \mathbf{K}_{ij} \mathbf{R}_b^T \tag{4.4}$$

$$\mathbf{K}_{ij}^{au} = \frac{V_a}{V} \mathbf{R}_a \Psi_{ai} \mathbf{K}_{ij} \mathbf{R}_a^T + \frac{V_b}{V} \mathbf{R}_b \Psi_{bi} \mathbf{K}_{ij} \mathbf{R}_b^T \tag{4.5}$$

$$\mathbf{K}_{ij}^{aa} = \frac{V_a}{V} \mathbf{R}_a \Psi_{ai} \Psi_{aj} \mathbf{K}_{ij} \mathbf{R}_a^T + \frac{V_b}{V} \mathbf{R}_b \Psi_{bi} \Psi_{bj} \mathbf{K}_{ij} \mathbf{R}_b^T \tag{4.6}$$

4.2.2 Stiffness Matrix Assembly

Since the tangent stiffness matrix is computed element-wise, it is necessary to assemble a global stiffness matrix $\mathbf{K}$. Assume there be n vertices in the tetrahedral mesh, the stiffness matrix $\mathbf{K}$ is a sparse matrix with the size of $3n \times 3n$. A vertex pair which belongs to the same tetrahedron will correspond to a 3×3 block of non-zero entries in $\mathbf{K}_e$. Since such a vertex part, which essentially forms an edge, is generally shared by multiple tetrahedra, the corresponding 3×3 block in in $\mathbf{K}$ is the sum of the per-element block from the respective $\mathbf{K}_e$. Therefore, the global stiffness matrix is a symmetric sparse matrix.

In the case of XFEM, we also need to take into consideration the arrangement of the added DOFs because of the cuts. In the following discussion, we follow our assumption in Chapter 3 that only a single cut exists within an element. However, it should be straightforward to extend our following discussion to a multiple cuts scenario.

Assume the deformable object is dissected by a splitting surface. The tetrahedra that have been intersected with the splitting surface in the tetrahedral mesh is then enriched using XFEM. Hence the splitting surface defines a global discontinuous function over the domain of all the enriched elements, and we have a set of enriched elements along with another set of original elements that are not enriched. From the set of enriched elements, we build up a set of enriched vertices which are the global added DOFs to the original system.

The key observation here is that there is no duplication of enriched vertices, i.e., each enriched vertex is listed only once even if it is shared by multiple enriched elements. This is illustrated in a 2-D example in Figure 4.2. The intuitive explanation of Figure 4.2 is that a cut only separates the enriched element into two parts, while both parts should be still connecting its own neighboring elements.

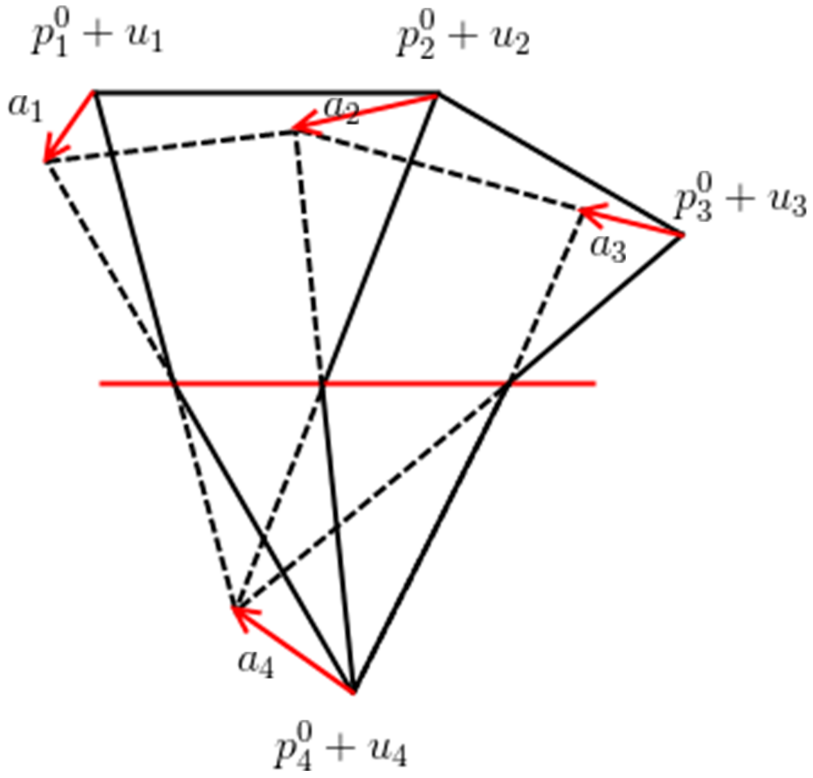


Figure 4.2: Shared Vertices Enriched Only Once for a Single Cut

As the result, we build up the set of enriched vertices $\mathbf{a} = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_{n^X}]$, where n^X is the number of enriched vertices. The global stiffness matrix therefore becomes:

$$\mathbf{K}^X = \begin{bmatrix} \mathbf{K}^{uu} & \mathbf{K}^{ua} \\ \mathbf{K}^{au} & \mathbf{K}^{aa} \end{bmatrix} \tag{4.7}$$

where $\mathbf{K}^{uu}$ is the original global stiffness matrix without enrichments, $\mathbf{K}^{ua}$ is a $3n \times 3n^X$ matrix, $\mathbf{K}^{au}$ is a $3n^X \times 3n$ matrix, and $\mathbf{K}^{aa}$ is a $3n^X \times 3n^X$ matrix. Conceptional, $\mathbf{K}^{ua}$ and $\mathbf{K}^{au}$ describe the Jacobian of the force on a non-enriched vertex with respect to the position of an enriched vertex. And similarly, $\mathbf{K}^{aa}$ describes the Jacobian of the force on an enriched vertex with respect to the position of another enriched vertex. The enriched global stiffness matrix is also symmetric just as $\mathbf{K}$, which gives $\mathbf{K}^{ua} = (\mathbf{K}^{au})^T$ and $\mathbf{K}^{aa} = (\mathbf{K}^{aa})^T$. Although we need to expand the global stiffness matrix when there are enriched elements, it is worth noting that the upper left sub-matrix $\mathbf{K}^{uu}$ remains unchanged. This is an important invariance during the simulation we can employ to speed up the simulation.

4.2.3 Mass Lumping

The mass matrix of the system is a symmetric $3n \times 3n$ matrix which describes the connection between the velocity field and the kinetic energy of an object. Assume the mass of a deformable object is continuously distributed, the consistent mass matrix is given as

$$\mathbf{M}_{ij} = \rho \int_V \phi_i \phi_j dV \tag{4.8}$$

The global mass matrix has the identical sparse structure as the stiffness matrix, since they both depend on the connectivity of the elements within the tetrahedral mesh. In a real-time application using FEM, we usually trade off accuracy for better computational performance by using a mass lumping method to construct the global mass matrix. The lumped mass matrix is a diagonal matrix constructed by summing up all entries of a row to the diagonal entry as

$$\overline{\mathbf{M}}_{ii} = \rho \sum_j \int_V \phi_i \phi_j dV = \rho \int_V \phi_i dV \tag{4.9}$$

Essentially, in a lumped mass matrix, the mass is concentrated at each node and therefore influence caused by the deformation of each element is ignored. It is an approximation of the consistent mass matrix, but it is easy to construct and most importantly, it is also easy to compute the inverse matrix.

As shown in [17], for an enriched element the mass matrix also has similar structure to Equation 3.19:

$$\mathbf{M}_e^X = \begin{bmatrix} \mathbf{M}^{uu} & \mathbf{M}^{ua} \\ \mathbf{M}^{au} & \mathbf{M}^{aa} \end{bmatrix} \tag{4.10}$$

where

$$\mathbf{M}_{ij}^{uu} = \rho \int_V \phi_i \phi_j dV \tag{4.11}$$

$$\mathbf{M}_{ij}^{ua} = \rho \int_V \phi_i \phi_j \psi_j dV \tag{4.12}$$

$$\mathbf{M}_{ij}^{au} = \rho \int_V \psi_i \phi_i \phi_j dV \tag{4.13}$$

$$\mathbf{M}_{ij}^{aa} = \rho \int_V \psi_i \phi_i \phi_j \psi_j dV \tag{4.14}$$

The lumped sub-matrices of $\mathbf{M}^{ua}$ and $\mathbf{M}^{au}$ are zero matrices, and there are different options of mass lumping method for $\mathbf{M}^{uu}$ and $\mathbf{M}^{a}$ that are discussed in Jeřábková's work [17]. In our system, we adapt the mass lumping method that is specific to XFEM

$$\overline{\mathbf{M}}_{ii}^{uu} = \frac{m}{n_e} \tag{4.15}$$

$$\overline{\mathbf{M}}_{ii}^{aa} = \frac{\phi}{n_e} \int_V \phi_i^2 dV \tag{4.16}$$

where m is the element total mass and n_e is the number of vertices within an element ($n_e = 4$ for tetrahedral meshes). The stability of this mass lumping method is analyzed in [17] as well.

The global mass matrix is assembled similarly as Equation 4.7:

$$\mathbf{M}^X = \begin{bmatrix} \mathbf{M}^{uu} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^{aa} \end{bmatrix} \tag{4.17}$$

where $\mathbf{M}^{uu}$ is the original global $3n \times 3n$ mass matrix and $\mathbf{M}^{aa}$ is a $3n^X \times 3n^X$ matrix.

4.2.4 Time Integration

To solve the system governed by Equation 4.1, our system uses the semi-implicit backward Euler method [23], which is unconditionally stable even for very stiff materials. Unconditional stability is essential in an interactive system. The robustness of such a system is not dependent on the time step and the stiffness of the simulated material. It guarantees that unpredictable user inputs would not accidentally destabilize the system. As pointed out by Baraff *et al.* [23], the semi-implicit backward Euler method tends to introduce excessive damping in large time-steps. But this drawback can be mitigated by picking appropriate time steps. Also, in the dissection simulation, the deformable object's motion is driven by the active external interactive forces. Damping is actually helpful to stabilize the system, and therefore the deformable object does not continue to bounce, even when there is no contact from the haptic tool.

Given the displacement field $\mathbf{u}_t$ and the velocity field $\mathbf{v}_t$ at time t, the integrator steps forward to $t + \Delta t$ and finds the displacements $\mathbf{u}_{t+\Delta t}$ and velocities $\mathbf{v}_{t+\Delta t}$ at the new time step. Let $\ddot{\mathbf{u}}_{t+\Delta t}$ be the acceleration fields at the end of the time-step, we have

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta t \mathbf{v}_{t+\Delta t} \tag{4.18}$$

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \mathbf{a}_{t+\Delta t} \tag{4.19}$$

Combined with Equation 4.1, we have

$$(\mathbf{M} + \Delta t \mathbf{D} + \Delta t^2 \mathbf{f}_{int}^{-1}(\mathbf{u}))\mathbf{a}_{t+\Delta t} = \Delta t(\mathbf{f}_{ext} - \mathbf{f}_{int}(\mathbf{u}_t) - (\Delta t \mathbf{K} + \mathbf{D})\mathbf{v}_t) \tag{4.20}$$

As we discussed above, the stiffness matrix and the lumped mass matrix are enriched accordingly in XFEM. We use Rayleigh damping where the damping matrix is a linear combination of the stiffness matrix and the lumped mass matrix:

$$\mathbf{D} = \alpha \mathbf{M} + \beta \mathbf{K} \tag{4.21}$$

Because of the stiffness matrix, the system governed by Equation 4.20 is a sparse system. Although it is possible to use a more computationally efficient direct solver to solve Equation 4.20, for large systems it is hard to fit the direct factorization into limited PC memory. In our system, we use conjugate gradient method to solve Equation 4.20 for $\mathbf{a}_{t+\Delta t}$. Although it is not as efficient as a direct solver, it is more memory efficient and is able to deal with large tetrahedral meshes. Also, the symmetry of the stiffness matrix is suitable for the conjugate gradient method.

4.3 Collision Detection

In an interactive dissection simulation, the deformable object and the instrumental tool (such as a scalpel) should interact with each other by some kind of physical contact. Collision detection is an essential component in an interactive simulation. It is used to (1) determine whether the deformable object and the instrumental tool are in contact, and if yes, (2) generate enough contact information (such as contact locations) in order to modify the state of the deformable object.

In our following discussion, we focus on the collision detection between a deformable object which is represented by a tetrahedral mesh and a haptic tool. We have two models for the haptic tool: (1) a simple point contact model represented by a sphere and (2) a cutting edge model represented by a line segment. The point contact model is a simplification of the "poking" operations in a dissection simulation, and the edge contact model is a simplification of the cutting front of a scalpel without considering the its exact geometry.

Collision detection itself is a broad research area that has many branches and applications. The full discussion of this field is beyond the scope of our thesis. We refer the interested readers to more specific literature in this field. Ericson [80] covers a broad area of collision detection techniques for rigid bodies including bounding volume hierarchies (BVH), spatial partitioning such as binary space-partitioning trees (BSP tree) that are viable in real-time applications. Teschner *et al.* [68] summarize state of the art collision detection methods specifically for deformable objects.

4.3.1 Surface Collision Detection

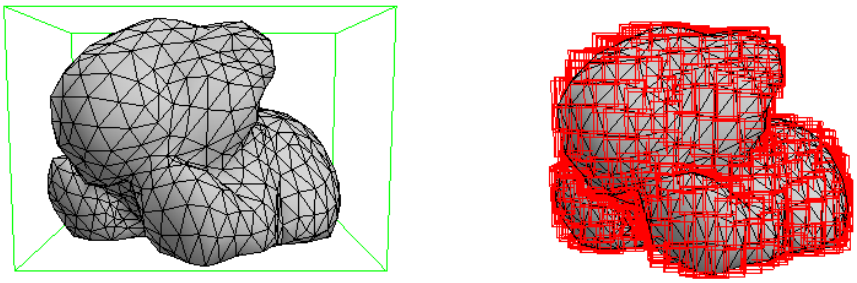
Surface collision detection is essential for both, the point contact model and the cutting edge contact model. In the scenario of non-invasive point contacts, surface collision detection is necessary to determine whether the haptic tool is in contact with the deformable object, and if yes, for finding out the contact location. In the cutting scenario, it is used similarly as in the point contact case to find the contact location on the object's surface and then to use this location as the starting point for searching the intersected elements along the cutting path.

Axis-aligned Bounding Box Hierarchy

In our system, the surface of the deformable object is generated from the underlying tetrahedral mesh. It is formed by the set of triangles which are not shared by multiple tetrahedra of the mesh. For a reasonable complex tetrahedral mesh, it is possible to have a surface mesh with thousands of triangles. Brute force collision detection between each triangle within the surface mesh and the haptic tool becomes computationally expensive in such scenarios. Also brute force is not scaling well in some complex scenarios such as collision between multiple deformable objects, which is very possible as a further work of our system. Hence, it is necessary to speed up the collision detection by avoiding brute force search.

As stated by Ericson [80], there are different options to speed up the collision detection. In our system, we construct a BVH over the surface triangles. BVH arranges the bounding volumes into a tree hierarchy to avoid unnecessary collision test. Each surface triangle is wrapped in a bounding volume, and the corresponding set of bounding volumes forms the leaf nodes of the tree. The leaf nodes are then grouped into small sets that are enclosed within larger bounding volumes which are in turn grouped together again recursively. This results in a tree structure with a root bounding volume which encloses all surface triangles. When performing collision detection on a BVH, there is no collision tests for child nodes if their parent fails the collision tests. Compared to brute force search, where the instrumental tool need to be tested against all surface triangles and therefore results in linear time complexity in the number of triangles, a BVH avoids unnecessary tests and reduces the time complexity to logarithmic time in the ideal case.

In our system, we use axis-aligned bounding boxes (AABB), the simplest bounding volumes, to construct the BVH. As mentioned in [80], there are more sophisticated options of bounding volumes, such as spheres, convex hulls, and discrete oriented polytopes (DOP). These more sophisticated options offer more strict results of the collision tests. However, the computation cost using them is generally more expensive than using the simplest AABB. Also, for deformable objects, the bounding volumes within their BVHs change frequently during the simulation. An AABB is very easy to update in such scenarios, while other bounding volumes may be more computationally expensive and hard to implement.

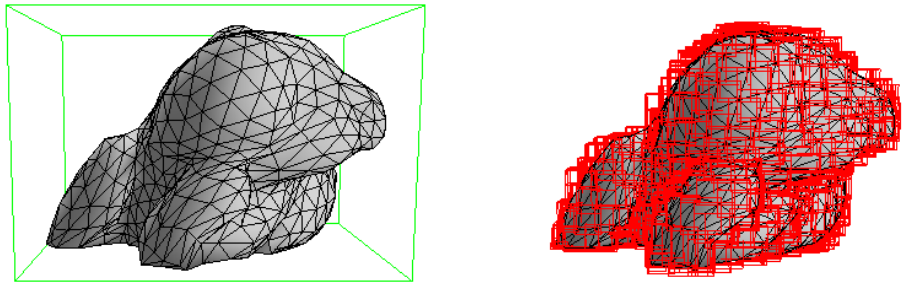


(a) The Root AABB in Green (b) The AABBs of Leaves in Red

Figure 4.3: The AABB Tree in the Reference Configuration

Updating Scheme

A deformable object may deform during the simulation, which is essentially equal to applying a displacement field over all vertices within the tetrahedral mesh representing this object. Therefore, the surface triangle mesh should deform accordingly to this displacement field. We use the method from van den Bergen's work [81] to update the AABB



(a) The Root AABB in Green (b) The AABBs of Leaves in Red

Figure 4.4: The AABB Tree in the Deformed Configuration

tree as the belonging object deforms. The AABB of each triangle is updated according to the displacements of its vertices, each parent AABB within the AABB tree is recomputed using the AABBs of its children using a bottom-up order. Similar to van den Bergen [81], we allocate an array of the references of the nodes within the AABB tree where the index of each internal child node in this array is greater than the index of its parent. Building this array can be done off-line by simply performing a breadth-first traversal over the original AABB tree. Then the whole process of refitting the AABB tree becomes iterating this array reversely and recomputing the AABB for each entry.

Because of the fact that the refitting method does not change the structure of the AABB tree, it is possible to cause a high degree of overlap as the deformation goes severe, as mentioned in [81]. However, van den Bergen claims that for deformations that keep the adjacency relation of triangles in a mesh intact, there is no significant performance deterioration for intersection tests [81]. Also, the performance overhead caused by rebuilding AABB trees, which has $O(n^2)$ time complexity, is hardly affordable for a real-time interactive simulation. By avoiding radical deformations such as excessive twists, the performance penalty from the overlap of the AABB can be avoided. By preserving the structure of the AABB tree during the simulation, we avoid the unnecessary performance overhead, which makes our project more feasible to low-end computers.

4.3.2 Volume Collision Detection

In a cutting scenario, it is necessary to detect the portion of an object that has been intersected with the cutting tool. We can model the path of the cutting tool between two consecutive time steps as a surface. Then we need to find out the elements within the tetrahedral mesh that are intersected with this cutting surface. Since there are interior elements that are intersected, the surface collision detection method is insufficient in this case.

We use a volume collision detection method which is derived from the method of Bielser and Gross [55] to find the intersected elements between two consecutive time steps. As mentioned by Bielser and Gross [55], the volume collision method exploits spatial coherency and involves local neighborhood search of adjacent tetrahedra. Our derived method is able to perform proper collision detection which is independent of the position of the cutting edge. Also, our algorithm is adapted to the XFEM framework, and we outline the necessary steps to enrich the intersected elements on top of the volume collision detection method.

The cutting tool is modeled as a line segment between the tool tip p_{tip} and the tool handle p_{hand}. Both p_{tip} and p_{hand} can be either inside or outside the deformable object. However, in reality, a dissection only begins when the cutting edge moves from the outside of the deformable object into the inside of it. Therefore, the surface collision detection is used to determine whether there is a collision between the line segment and the deformable object, and if yes, it finds the contact location on the surface mesh and proceeds to our volume collision detection.

We use the same notion of *active tetrahedra list* as Biesler *et al.* [54, 55], which is a list of tetrahedra that has at least one intersection point with the cutting tool. We construct the initial active tetrahedra list by performing a neighborhood search starting from the surface tetrahedron which is detected by the surface collision detection (see Algorithm 1). Assume tet_{start} to be the boundary tetrahedron that contains the contacting surface triangle from the surface collision detection. The neighborhood search starts with the surface tetrahedron which is detectd in the surface collision detection, which gives $tet_{prev} = -1$ and $tet_{curr} = tet_{start}$ to the Procedure **FindInitTet**. All active tetrahedra are then detected in the neighborhood search and added into the **currActiveTetraList** during this recursive process.

Algorithm 1 Find Initial Active Tetrahedra List

```
procedure FindInitTet(tet_prev, tet_curr, p_tip, p_hand)
    currActiveTetraList appends tet_curr
    for all face ∈ tet_curr do
        if Intersect(face, p_tip, p_hand) then
            tet_next ← neighbor of face
            if tet_next ≠ tet_prev then
                FindInitTet(tet_curr, tet_next, p_tip, p_hand)
            end if
        end if
    end for
end procedure
```

We denote t_n as the current time frame, and t_{n+1} as the next time frame. Between any two consecutive time steps, the positions of the line segment in the two respective time frame form a splitting surface. We need to find the tetrahedra that have been intersected with the splitting surface. Algorithm 2 outlines the neighboring search performed on a tetrahedron. It is applied on every tetrahedron that is listed in the **currActiveTetraList**, and then recursively searches the neighborhood of a tetrahedron. To avoid infinite loops in

this recursive algorithm, we set up a Boolean flag **isVisited** for each tetrahedron so that each tetrahedron is only tested once between two time steps.

It is also worth noting that the **Process** function can be a highly-customized function manipulating the states of the intersected edge. In our project, we limit the number of cuts within an element to one. Our **Process** function simply tests if the edge is intersected before, and if not, records the ratio of the intersection point and sets the Boolean flag **isIntersected** for this edge. However, it is possible to allow multiple cuts against an edge and the **Process** function can be highly customized. Therefore, we outline this function in our pseudo code as an abstraction of the potential process to deal with the edge intersections.

Algorithm 2 Find Intersected Tetrahedra

```
procedure FindInterTet(surface, tet)
    if tet.isVisited = false then
        currIntersectedTetraList appends tet
        tet.isVisitied ← true
    end if
    for all edge ∈ tet do
        if Intersect(surface, edge) ∈ (0, 1) then
            Process(surface, edge)
            for all tet_next ∈ neighbor of edge do
                FindInterTet(surface, tet_next)
            end for
        end if
    end for
end procedure
```

Algorithm 3 Process Intersected Edge

```
procedure Process(surface, edge)
    if edge.isIntersected = false then
        edge.intersectionRatio ← Intersect(surface, edge)
        edge.isIntersected ← true
    else
        PotentialManipulation(surface, edge)
    end if
end procedure
```

After we find out the intersected tetrahedra, it is necessary to find the active tetrahedra list at t_{n+1}. In [55], there is an assumption that the tool handle p_{hand} is always outside of the deformable object. This a reasonable assumption in reality because in rare cases the user's hand holding the cutting handle may enter the interior of the deformable object. But we believe a more general dissection without this assumption would be helpful because of the unpredictable user inputs in a haptic simulation. It is possible that the handle of

the cutting tool enters the interior of the deformable object while the user does not notice. We use a more general method so that the simulation continues to work even if this case happens during the simulation, which improves the robustness of our system.

The starting point for finding active tetrahedra for the next frame is that the set of active tetrahedra is a subset of the intersected tetrahedra. After we find the intersected tetrahedra, we test them against the current position of the cutting tool to determine if they are still intersected by the cutting tool. This forms the active tetrahedra list for next time frame, which is then used as the seed to find the intersected tetrahedra in the next time interval.

Algorithm 4 Find Active Tetrahedra List

```
procedure FindNextActiveTet(p_tip, p_hand)
    clear currActiveTetraList
    for all tet ∈ currIntersectedTetraList do
        if Intersect(tet, p_tip, p_hand) then
            currActiveTetraList appends tet
        end if
    end for
end procedure
```

The overall work-flow of our collision detection is depicted as Algorithm 5. The first part of the algorithm uses the surface collision detection method to determine if the cutting tool is in contact with the deformable object, and if yes, then we switch to the volume collision detection. It is still worth noting that the position of the handle needs to be outside of the deformable object when the dissection process begins. However, this assumption is still more flexible than at all times as in [55], and it is reasonable in reality. The second part is the workflow of the volume collision detection using Algorithm 2 and 4. The splitting surface is approximated by two triangles formed by the current positions (p_{tip} and p_{hand}) and the previous positions (p^0_{tip} and p^0_{hand}) of the cutting tool. When the size of **currActiveTetraList** becomes zero, this dissection process is finished and the Boolean flag **isCutting** is set to $false$ again. The collision detection then is switched back to the surface collision detection state.

4.4 Haptic Rendering

The collision detection process provides enough information about the collision between the deformable object and the instrumental tool. The collision is then resolved by the physics engine that generates appropriate responses to both colliding objects. As we have mentioned earlier, the collision response of deformable objects during dissection is still an active research area and it is beyond the scope of our work. However, we want to incorporate the haptic rendering for point contacts to provide enough immersion. And it is an essential feature in a dissection simulation that allows users to explore the physical properties of the deformable object by performing non-invasive contacts (i.e, poking) and

Algorithm 5 Volume Collision Detection

```
if (isCutting = false and surface.isContacted = true) then
    tet_begin ← tetrahedron contains surface.contactFace
    isCutting ← true
    FindInitTet(−1, tet_begin, p_tip, p_hand)
end if
if isCutting = true then
    surface ← MakeSurface(p^0_tip, p^0_hand, p_tip, p_hand)
    for all tet ∈ tetrahedralMesh do
        tet.isVisited ← false
    end for
    for all tetra ∈ currActiveTetraList do
        FindInterTet(surface, tet)
    end for
    FindNextActiveTet(p_tip, p_hand)
    if currActiveTetraList.size = 0 then
        isCutting ← false
    end if
end if
```

then observing the deformation. Our goal is to provide a fast and robust haptic rendering for these point contacts to provide better interactive immersion to users without sacrificing the performance of the simulation.

4.4.1 God-object Algorithm

From the surface collision detection, we are able to obtain enough collision detection information of a point contact. However, we need a collision resolution method for non-invasive point contact The tip of haptic tool, which also referred as *haptic interaction point* (HIP) in such cases, must be inside the deformable object when they are in contact, but in a non-invasive scenario we want to constrain the HIP to be always on the surface of the deformable object. The easiest method is to attach a stiff spring to the HIP when it penetrates the surface of an object. Then a large force is caused by the penetration and the HIP is pushed towards the surface. However, many studies have shown that this causes stability issues that impairs the immersion during haptic rendering.

There are constraint-based methods to deal with non-penetrating haptic rendering for point contacts such as God-object algorithm [70] and the Virtual Proxy algorithm [71]. We use the God-object algorithm [70] in our system. The notion of "God-object" describes the virtual location of the HIP on the object's surface. The location of God-object is computed to be a point on the current contacting surface such that the its distance from HIP is minimized [70]. By constraining the location of the God-object to be always on the surface, we give users a feeling that the object is in-penetrable. And the distance between the HIP and the God-object is also helpful to apply penalty-based haptic rendering. In

the next section, we use this distance to compute the haptic force response based on the local material properties.

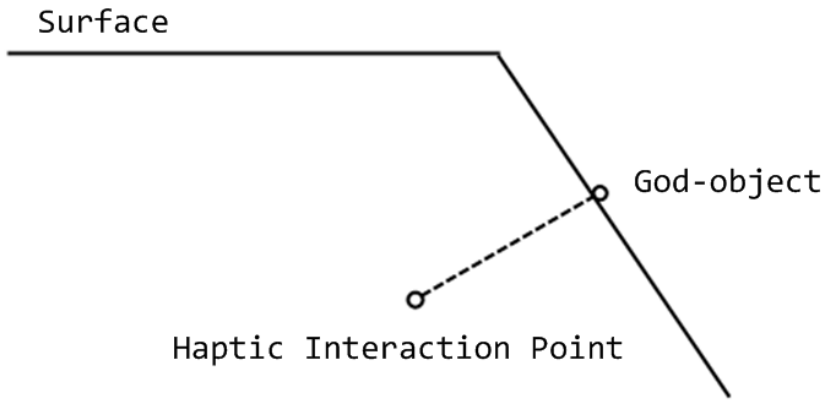


Figure 4.5: God-Object in a Non-Invasive Point Contact

4.4.2 Local Force Computation

From the God-object algorithm, we compute the distance between the HIP and the God-object, which is the virtual position of the HIP on the object's surface. This distance is also referred as *penetration depth* in penalty-based methods. The force response computed from penalty-based methods is intuitively analogous to Hooke's law $f = kx$, where k is the stiffness constant and x is the extension of the spring.

In our case, the stiffness of a tetrahedron is a 12×12 matrix. We denote the penetration depth as $\mathbf{d}$. The force response of the non-invasive point contact is computed so that the contacting tetrahedron would push the HIP out and make $\mathbf{d} = \mathbf{0}$. This means the expected displacements of each vertex in the contacting tetrahedron is $\mathbf{d}$. The local force response caused by the non-invasive contact is given as

$$\mathbf{f}_{local} = \mathbf{K}_e \mathbf{u}_d \tag{4.22}$$

where $\mathbf{u}_d = [\mathbf{d}\ \mathbf{d}\ \mathbf{d}\ \mathbf{d}]$ is the concatenated vector of expected displacements, and $\mathbf{K}_e$ is the stiffness matrix of the contacting tetrahedron. In the co-rotational FEM method, we also

need to take into consideration the rotation of the contacting element, which gives

$$\mathbf{f}_{local} = \mathbf{R}_e \mathbf{K}_e \mathbf{R}_e^T \mathbf{u}_d \tag{4.23}$$

where $\mathbf{R}_e$ is the rotational matrix of the tetrahedron.

The local force computation should be enough for a rigid-body of which the shape does not change even if a contact force is applied. However, the contact force would compress the volume of the deformable object and therefore penetration depth between the surface and the HIP is not as large as in rigid-body cases. Hence the user feels the deformable object is too "soft" because the magnitude of the local force computed by Equation 4.22 or 4.23 is small.

We compensate this by adding the global deformation force to the haptic force. For a deformed object, the deformation force always tries to pull it back to its reference configuration. Therefore, for a compressed deformable object, we can compute a global force factor by interpolating the internal forces of the contacting tetrahedron based on the barycentric coordinate of the God-object.

$$\mathbf{f}_{global} = \sum_{i=1}^{4} \Phi_i \mathbf{f}_{int}^i \tag{4.24}$$

The actual haptic force is the sum of the local penetration force and the global deformation force. However, it is important to figure out whether the deformation force should be added to the haptic force. Figure 4.6 and 4.7 illustrate two different cases of the global deformation force. And in Figure 4.6, the user is actually not touching the compressed side of a deformed tetrahedron. If we add the deformation force into the haptic force, the user would feel that the HIP is being absorbed into the deformable object, which is definitely not a realistic feeling of touching an object. Hence, we only add the global deformation force into the haptic force when the angle between the local force and the deformation force is less than 90°; otherwise we only rendering the local penetration force to the haptic device (See Equation 4.25, Figure 4.6 and 4.7).

$$\mathbf{f}_{haptic} = \begin{cases} \mathbf{f}_{local} + \mathbf{f}_{global} & \text{if } \mathbf{f}_{local} \cdot \mathbf{f}_{global} > 0 \\ \mathbf{f}_{local} & \text{if } \mathbf{f}_{local} \cdot \mathbf{f}_{global} \leq 0 \end{cases} \tag{4.25}$$

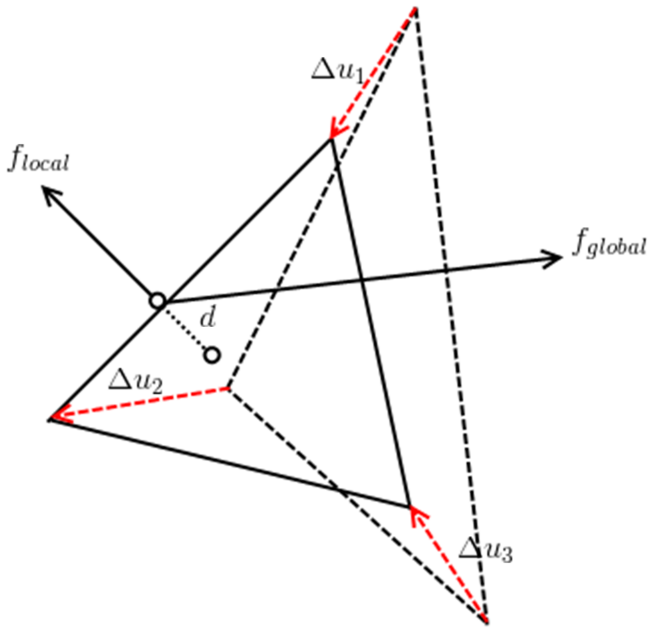


Figure 4.6: The Haptic Force Not Consisting of the Global Deformation Force

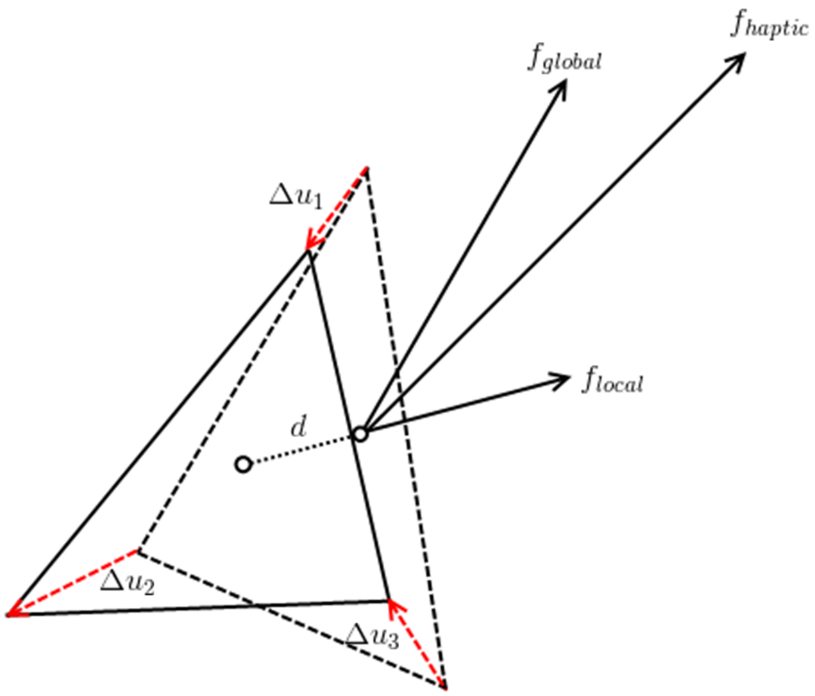


Figure 4.7: The Haptic Force Consisting of the Global Deformation Force

Chapter 5

Implementation

In this chapter, we provide a detailed review of our implementation, which employs several open-source components to ease the development. Section 5.1 provides the implementation of each component. We analyze the needs of our system and outline how an open-source component can benefit our development. We also demonstrate an interface-based design of the interaction between the haptic tool component and the deformable object component which does not rely on implementation details and therefor is extensible to supports new features. The synchronization scheme between the haptic thread and the dynamic thread is also demonstrated.

5.1 Implementation Designs

In an object-oriented design, the implementation of an interactive simulation system consists of two essential components:

- the deformable object class managing the dynamic states, and
- the haptic tool class interfacing the haptic devices and the virtual scenario.

Building all these components from scratch requires a great amount of efforts as well as a broad knowledge of all these fields. Fortunately the open-source community has joined together by offering useful middle-ware libraries which make the implementation much easier. Our system leverages several open-source libraries including Chai3D [4], OpenVolumeMesh [18], Bullet [19] and Vega FEM [20]. By using existing libraries, we avoid reinventing the wheel and focus on the general design of the interactive system. Figure 5.1 illustrates the dependency structure of our system.

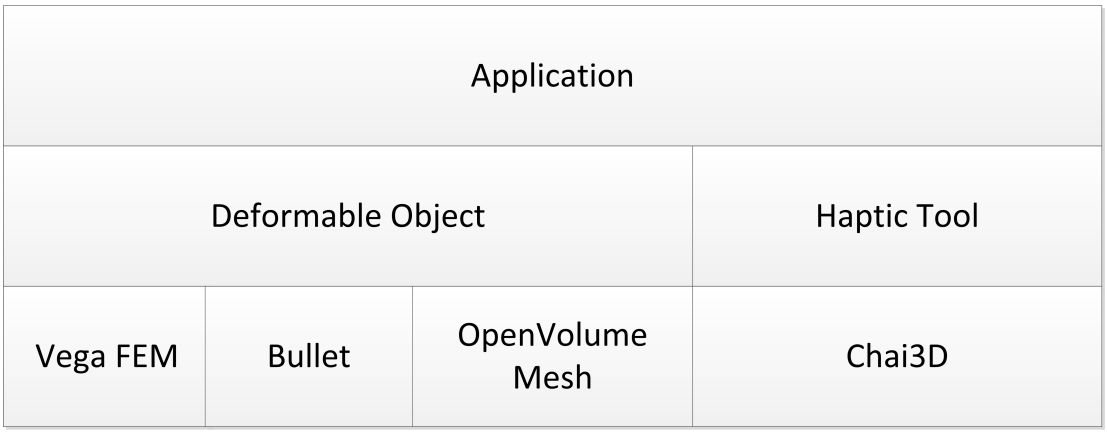


Figure 5.1: The Dependency Structure of an Interactive Dissection Simulation

5.1.1 Haptic Tool

Chai3d

Chai3D is an open-source multi-platform haptic rendering framework written in C++ [4]. It provides a set of useful functionality including

- graphics rendering with scene graph support,
- haptic rendering algorithms for rigid bodies,
- collision detection algorithms for rigid bodies,
- an universal haptic device interface, and
- I/O utilities

to ease the implementation of a simulation with haptic support. It is also extensible to other third party components through the extension modules. This make Chai3D very suitable for prototyping research projects.

Our system leverages the universal haptic device interface provided by Chai3D to support a broad range of haptic devices, including the Novint Falcon® and the Geomagic® Touch™ (formerly Sensable Phantom Omni). By using Chai3D, our system is independent of the actual haptic device and it is possible for the user to choose the appropriate device based on their needs and budget. Also, it is possible to integrate custom device using the template provided by Chai3D. The users are given a great degree of freedom in selecting the haptic devices, which makes our system more feasible to a broader area including general education.

The abstraction of the haptic device hardware using the universal haptic device interface is also helpful to the implementation. Every manufacturer of haptic devices has its own Software Development Kit to program the haptic hardware. It is hard to implement a

general haptic tool that relies on the actual APIs of different existing SDKs. By using Chai3D, we decouple the implementation of our haptic tool from the SDK coming from haptic devices. It is possible to abstract and implement haptic rendering and collision detection algorithms in an universal way without knowledge of a specific SDK of an haptic device. And it is easy to share our system on other platforms which may use different hardware or operating system.

Design Details

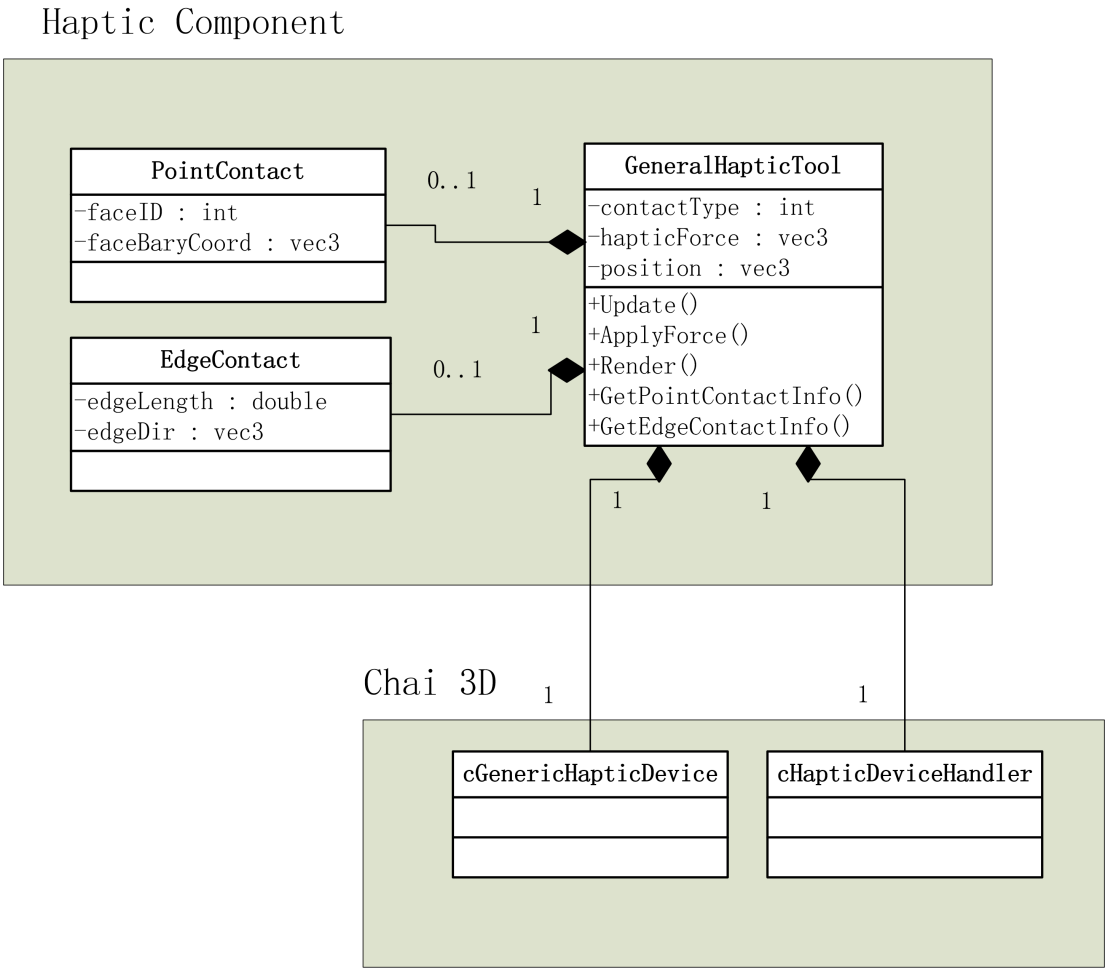


Figure 5.2: The Design of the Haptic Tool Class

Figure 5.2 illustrates the design of the **GeneralHapticTool** in our system. It is worth noting that **GeneralHapticTool** has no knowledge of any APIs of haptic device SDK. It leverages the **cHapticDeviceHandler** and **cGenericHapticDevice** to interface the underlying haptic device. They provide a set of universal interfaces to set and query states of the underlying haptic device. In our system, our haptic tool can switch between point contact and edge contact. The **PointContact** and **EdgeContact** are used to describe the geometric properties of the contact. Although we store the position of the HIP in

the **cHapticDeviceHandler**, since our system uses a God-object based method for non-penetration point contact, we need to store the information of the corresponding God-object in **PointContact**. However, storing the absolute position of the virtual proxy is problematic because it is not guaranteed to be on the object's surface as the object deforms. As the result, **PointContact** keeps track of the ID of the contacting surface triangle and the barycentric coordinate of the virtual proxy within the triangle instead.

5.1.2 Deformable Object

The deformable object class is the core component of our system which manages the whole dynamic information of a deformable object during the simulation. The dynamic information primarily consists of (1) the deformation states and (2) the collision information. 5.3 illustrates the design of the **DeformableObject** class based on several open-source middleware libraries including Vega FEM, OpenVolumeMesh and Bullet. In the following, we present a brief review of each open-source library and outline its usage in our system. Then we detail the necessary design considerations of implementing the XFEM based on these components.

Vega FEM

Vega FEM is open-source middle-ware physics library for 3-D deformable object simulation [20]. It implements a wide range of deformable models including a mass-spring system, co-rotational linear FEM, Saint-Venant Kirchhoff FEM and invertible non-linear FEM. It also provides different options for integration including implicit backward Euler and Newmark methods which are appropriate for stable deformation simulations. It is written in C++ with minimal dependency on other libraries, which make it appropriate to be integrated into our system.

The design of Vega FEM decouples the elastic force computation of a deformable model from the integrator implementation. An abstract class **ForceModel** provides an universal interface of elastic force computations which is required by the integrator. Then the actual implementation of a deformable model is independent of the integrator, which makes it easy to support different deformable models within the same framework. Also, the implementation classes of the integrators are derived from an abstract class **SparseBaseIntegrator**. Therefore, it is possible to change the underlying integration method without modifying the application code.

Bullet

Bullet is an open-source physics library for real-time physics simulations [19]. The main features of Bullet includes 3-D collision detection and rigid body dynamics. Also, the support of deformable dynamics is a feature in progress. It is a middle-ware written in C++ and has been used in many games and movies.

We do not adapt the whole physics pipeline of the Bullet engine into our system. Instead, we only use some of the components to help implementing the **DeformableObject** class. The vector math library of Bullet, which provides a set of efficient implementations of linear algebra functions, is used in our system. Also, we adapt the dynamic AABB tree implementation (**btDbvt**) in the surface collision detection of our system. **btDbvt** is a dynamic general purpose AABB tree that features fast insertion, removal and updates. In the point contact scenario, the object is tested against the line segment between the previous God-object location and the current HIP location. Also, the edge contact requires the collision test between the edge and the deformable object. Therefore, we only need to implement one query "collideRAY" of the **btDbvt**. The **DeformableObject** class stores its own AABB tree as a private data member.

OpenVolumeMesh

OpenVolumeMesh is an index-based generic data structure for handling arbitrary polyhedral meshes [18]. It has the similar concept to OpenMesh [82], which implements the generic half-edge data structure where each edge of a polygonal mesh is split into two oriented parts (i.e. half-edges) to store the connectivity information. OpenMesh is used to handle two-manifold geometric meshes and is not suitable for volumetric meshes (three-manifold). OpenVolumeMesh carries the general idea by splitting each face in a polyhedral mesh into two oriented half-faces that have opposite orientations. By storing the incident polyhedron (which is also called *cell* in OpenVolumeMesh) for each half-face, it supports a wide range of local query operations on the meshes. For example, it provides **HalfEdgeCellIter** which iterates over all incident cells of a given half-edge.

The local query operations are very helpful in the implementation of our volume collision detection. In Algorithm 1 we need to find out the neighboring tetrahedron of a face. And in Algorithm 2 we need to find out the neighboring tetrahedra of an intersected edge. Therefore, we use OpenVolumeMesh in the implementation of our **DeformableObject** class to store the connectivity information and then use it in our volume collision detection.

Design Details

Figure 5.3 illustrates the designs of the **DeformableObject** class in our system. As we mentioned above, it consist of two main tasks: (1) the collision detection and (2) the dynamic deformation. The **CheckContact** function is responsible to detect whether the deformable object has collided with the haptic tool. If it is in a point contact scenario, this function also resolves the collision and generate the interactive force that is applied to the deformable object. It is worth noting that this interfacing function does not impose any restriction on the underlying collision detection implementation. As shown in Figure 5.2, the haptic tool provides the required contact information to the deformable object, and it is extensible to provide additional information based on the user's need. The details of the collision detection is encapsulated inside this function. It is possible to use different collision detection methods and different data structures to implement this function while not changing the application code, which gives great flexibility to the system.

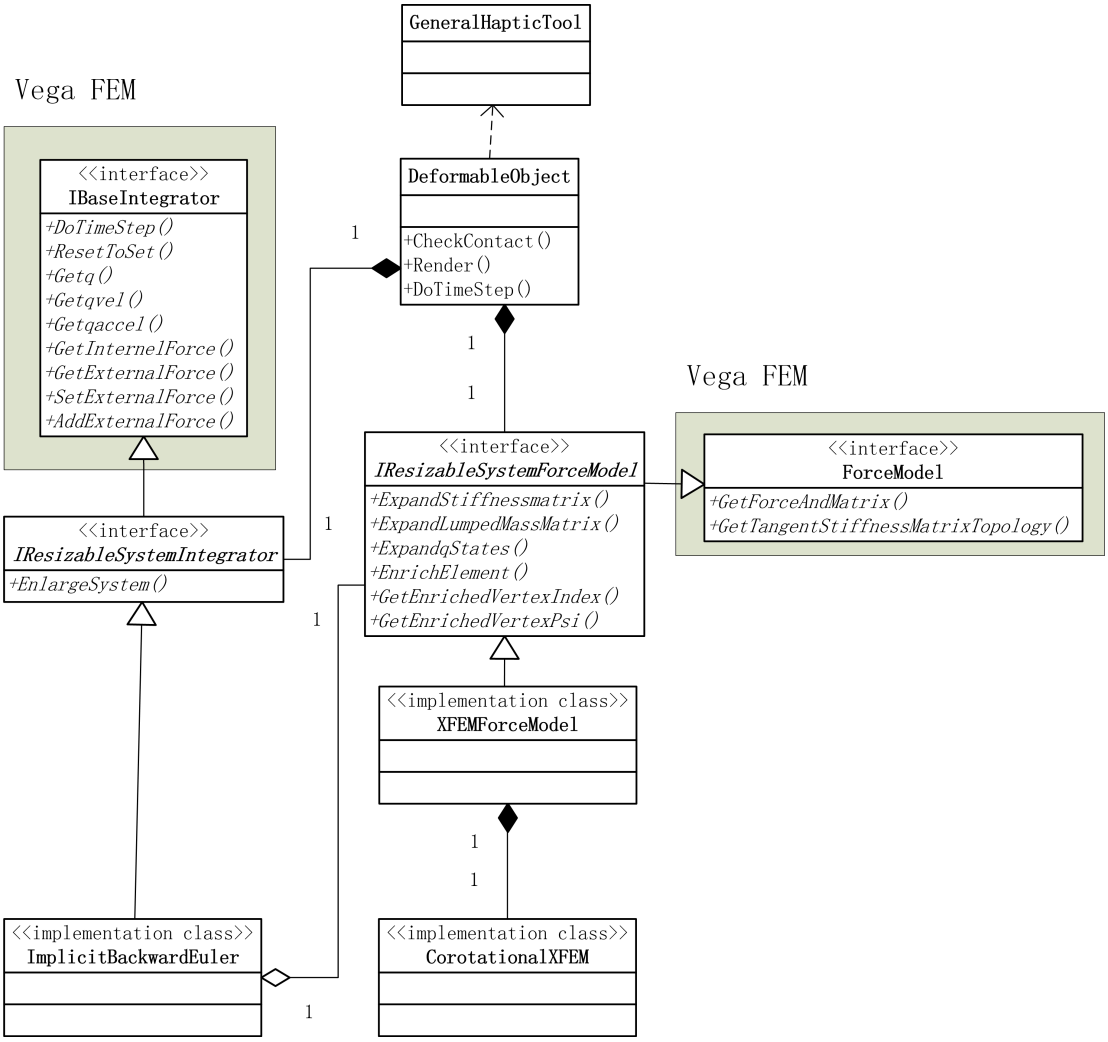


Figure 5.3: The Design of the Deformable Object Class

The design of the dynamic deformation component is based on the **ForceModel** and **SparseBaseIntegrator** interfaces of the Vega FEM library. However, the Vega FEM does not incorporate any collision detection mechanism in their library. Also, it is not designed for dissection simulation, which means the deformable models and the integrators do not provide any functionality to describe cuts. Our implementation extends the Vega FEM to support dissections by designing a set of derived interfaces.

The key functions supported by the abstract **ForceModel** are listed in Figure 5.3. In the design of Vega FEM, the functions interface to the required operations by an integrator. However, in a dissection simulation, the function of a deformable model needs be extended beyond the elastic force computation. The **IResizableSystemForceModel** interface abstracts the necessary operations in a dissection simulation based on the XFEM framework (enrichment functionality without topology changes). To be able to introduce cuts, it is necessary to enrich elements and store the corresponding information in the deformable model. The abstract virtual function **EnrichedElement** enriches an element with the

discontinuous information, which is typically provided by the collision detection in a dissection simulation. As we discuss in Chapter 3, the stiffness matrix and the lumped mass matrix are subject to change during the dissection simulation. Because the enrichment information of elements are stored in the deformable model while the system matrices are stored in the integrator, it is therefor necessary to provide the matrix expansion functions in the deformable model which can be called by the integrator. Given the previous global stiffness matrix, **ExpandStiffnessMatrix** expands it to the enriched stiffness matrix as in Equation 4.7. **ExpandLumpedMassMatrix** is essentially the same except that the lumped mass matrix can be represented by an array.

Similarly, the **IResizableSystemIntegrator** interface extends the original **IBaseIntegrator** interface by allowing the system expansion on the fly. The **CopyStiffnessMatrix** function takes an enriched stiffness matrix as an input, and updates the system stiffness. The **EnlargeSystem** function extends the system of the integrator by enriching the stiffness matrx, the mass matrix and the damping matrix. Also, the vectors of displacement, velocity and acceleration need to be re-allocated as well.

In the XFEM framework, the main expense caused by dissections is basically the time spent on expanding the stiffness matrix of the deformable object. The stiffness matrix generally has a sparse structure which is dependent on the connectivity of the vertices in the tetrahedral mesh. In our system, we use Vega's sparse matrix implementation, which use the compressed sparse row format to store the matrix. In order to find the correct mapping between an entry of an element stiffness matrix and an entry of the global stiffness matrix, the indices of of non-zero entries in each row are pre-sorted at startup to avoid real-time computational cost [20]. A key observation of the global stiffness matrix (Equation 4.7) is that the added DOFs are appended incrementally. The global stiffness matrix before an enrichment becomes the upper-left sub-matrix, and its internal structure remains unchanged after the enrichment. As the result, the previous stiffness matrix as well as the pre-sorted indices are able to be re-used after the stiffness matrix has been enriched. We only need to append newly added DOFs to the stiffness matrix, and append new indices for these DOFs accordingly. The computational overhead of rebuilding the sparse matrix and the indices is therefore avoided.

5.1.3 Synchronization Scheme

As we mention before, our system consists of two threads that run at different rates. It is necessary to design a proper synchronization scheme so that the system will not run into multi-threading issues such as a deadlock or a race condition. Figure 5.4 illustrates the synchronization scheme we use in our system.

Generally, it takes much more time to step forward in the dynamics thread in comparison to the haptic thread which only involves the collision detection and the local force computation. It is definitely problematic if the haptic thread modifies the contact information during the dynamic time stepping is in progress. Also, it is possible that the dynamic thread changes the displacement field when the collision detection is in progress. Both cases would at least lead to wrong results and even worse, destabilize the simulation.

The main idea of our design is that we assume the dynamics of the deformable object does not change when the dynamic time stepping is in progress. In order to do this, the haptic thread keeps its own copy of the displacement field and of the stiffness matrix which are required for collision detection and haptic force rendering. These data are synchronized from the result of the last dynamic time step and they will not change until the current time step is finished. Also, before the beginning of a time step in the dynamic thread, the external force or the intersected elements are synchronized from the haptic thread. Hence, there are two synchronization locks required at the beginning and the end of a dynamic time step.

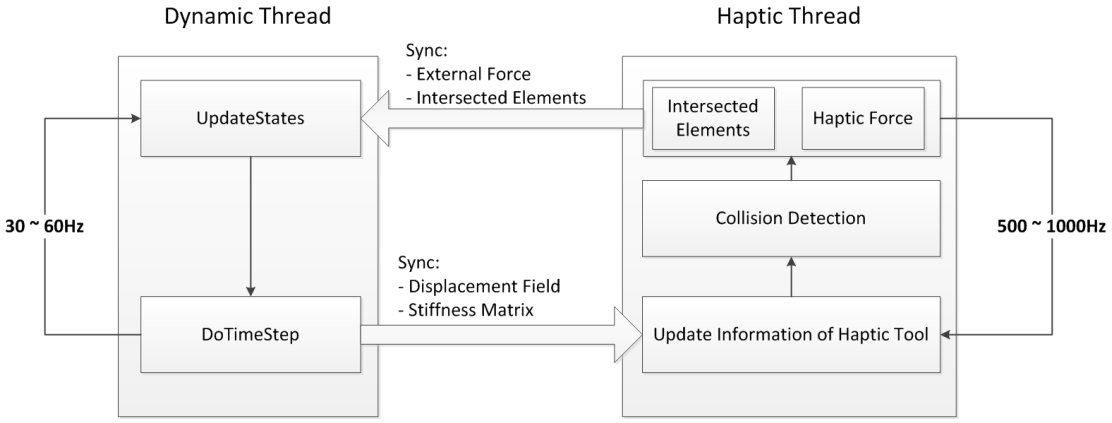


Figure 5.4: Synchronization Scheme between the Dynamic Thread and the Haptic Thread

Chapter 6

Experimental Evaluation

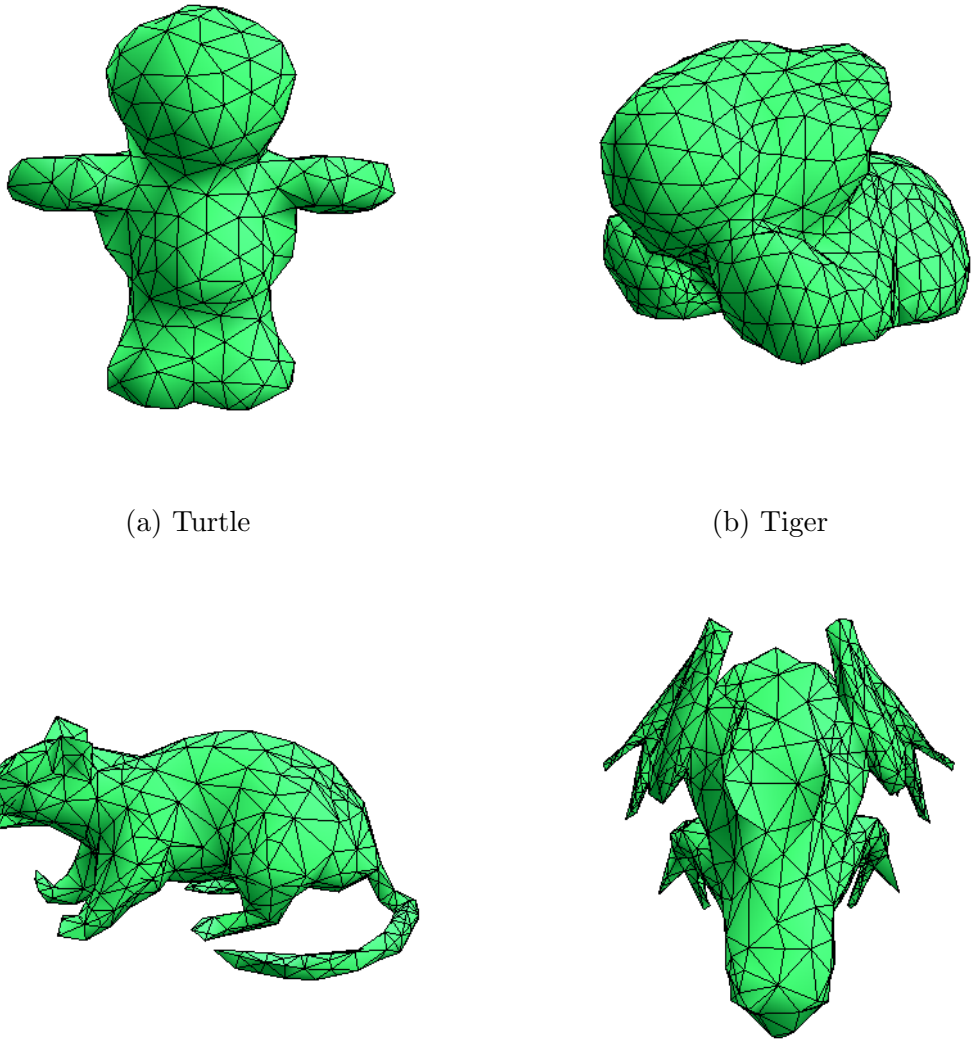
The experiments are performed on consumer PC with an Intel four-core 3.40 GHz i7 CPU and 16 GB RAM running Windows 7. The PC is also equipped with a Sensable Phantom Desktop as the haptic interface to the user. Our system is tested on several tetrahedral meshes that have different shapes and resolutions (see Table 6.1 and Figure 6.1). The turtle mesh was created by Jeremy Quandt, who also owns copyright on it. It was downloaded from `http://www.blendswap.com`. The volumetric mesh and its surface mesh were both created by Jernej Barbič and released with Vega FEM. The rat and frog meshes were downloaded from Autodesk 123D and processed by Meshlab. To better compare the performance, we use the same homogeneous material for all meshes. The material is parameterized by Young's modulus (E) and Poisson's ratio (ν). In our experiments, we specify a material with $E = 0.1\ MPa$, $\nu = 0.4$ and mass density $\rho = 1000\ kg/m^3$.

Mesh	Turtle	Tiger	Rat	Frog
#Vertices	347	619	629	826
#Elements	1185	1904	1961	2301

Table 6.1: Resolution of Sample Meshes

6.1 Integration Time

One of the reasons that we build our system on top of Vega FEM is because it offers a large set of C++ implementations of different material models and integration schemes. The integration time of the original Vega FEM is used as a benchmark for our additions and modifications. By comparing the performance of our system, which is developed on top of Vega FEM, and that of the original implementation, we can analyze the characteristics of our system. Here we are only looking at the performance of our method for deformation, and therefore there is no dissection added to the deformable object. In order to avoid visual artifacts in large deformation scenarios, we use the co-rotational linear FEM to model



(a) Turtle (b) Tiger

(c) Rat (d) Frog

Figure 6.1: Example Meshes

deformations. We test the integration time of a single step on each of the tetrahedral mesh in Table 6.1. The Jacobi-preconditioned conjugate gradient (PCG) solver of Vega uses zero as an initial guess and stops when the residual norm drops below $\epsilon = 10^{-6}$ or the maximum number of iterations $N_{max}$is reached. We use $N_{max} = 10000$ as the maximum number of iterations in all our experiments. It is worth noting that we set up a strict convergence criteria in PCG, but it is possible to trade in the precision for better efficiency by modifying ϵ and N_{max}. All the operations apart from rendering are performed using a single core.

The experimental results marked with "Vega" are tested with the original implementation of the implicit Euler integrator and the PCG solver [20]. Since our system requires that

the integration, the sparse matrix and the PCG solver must be resizable as we explain in Section 5.1.2, our system cannot use Vega FEM unaltered. Therefore, the performance of Vega FEM is reported from a separate application which does not have dissection support. Those marked with "XFEM" are tested with our modified version of the sparse matrix of the PCG solver and of the implicit Euler integrator. Since we are not introducing dissections in this experiment, the main difference between Vega FEM and our implementation is how a sparase matrix copies values from another matrix of possibly different size.

The average times in milliseconds are shown in Table 6.2. It shows the performance of our modified implementation is close to the original Vega FEM (marked by "Vega" in Table 6.1). In fact, we observe that our implementation is slightly more efficient in most scenarios. The total dynamics time is roughly the sum of the force assembly time and the system solving time. It can be observed that the time of force assembly only takes a small portion of the total dynamics time at each timestep. Because we use the implicit Euler integration in our system, solving the linear system becomes the most computationally expensive operation at each time step. As we mention above, we use a strict convergence criteria in the PCG, therefore the number of iterators range from 80 to 300 in different scenarios. We obsever the total time of solving the system is proportional to the number of required iteration of the PCG solver. It is also reported by Sin *el at.* [20] that the number of required PCG iterations is proportional to the stiffness of a object. Therefore, the total time for solving the linear is dependent on the number of iterations required by the PCG solver. It is possible to set a smaller value of the maximum number of iterators in order to obtain better update rates.

Computation Step	Sample Meshes			
	Turtle	Tiger	Rat	Frog
Timestep (msec)	10	40	20	100
Vega: Total Dynamics (msec)	8	47	16	82
Vega: Force Assembly (msec)	2	4	6	4
Vega: System Solve (msec)	5	42	9	76
Vega: #Iterations	46	226	42	312
XFEM: Total Dynamics (msec)	7	43	16	91
XFEM: Force Assembly (msec)	2	4	4	5
XFEM: System Solve (msec)	4	38	10	84
XFEM: #Iterations	38	215	50	320

Table 6.2: Average Breakdown of Integration Times. Shown are results for unaltered Vega and our XFEM extension.

We also compare the performance of our system with the work of Jeřábková [83]. They only provide the numbers for deformation and not for cutting in their experiments. They report the performance of their system on a bar mesh with 12800 tetrahedral elements and 3321 vertices using the co-rotational FEM. The material is parameterized by $E = 0.1\ MPa$, $\nu = 0.33$ and $\rho = 100\ kg/m^3$. In the experiment, implicit Euler integrator is used for time stepping, and 10 conjugate gradient iterations are performed at each time. A total time of

5 seconds is simulated with a 40 ms time step. In their single-threaded implementation, it takes 36.5 seconds to finish the 5 seconds of simulation, which is not close to our real-time requirement. However, Jeřábková reported this result in 2007 and their numbers would likely by more favorable today.

In order to compare with Jeřábková, we test the performance of our system on a bridge mesh of very simular size than their bar mesh. The bridge mesh was created by Jernej Barbič and released with Vega FEM (see Figure 6.2) and it has 4000 vertices and 12827 elements. The material is specified using the same parameters ($E = 0.1\ MPa$, $\nu = 0.33$ and $\rho = 100\ kg/m^3$). The maximum number of CG iterations N_{max} is set to 10 as in the work of Jeřábková [83]. For a real-time application, the integration time must be smaller than the actual timestep in the simulation. In our experiment, the size of a time step is 70 ms, and the average integration time of the mesh is 63 ms over 400 time steps. Our system is therefore able to achieve about 15 frames per second in real-time, which is sufficient for interactive applications. This shows that the computational power of a normal PC is sufficient to perform interactive simulations that were supposed to run on a high-end workstation.

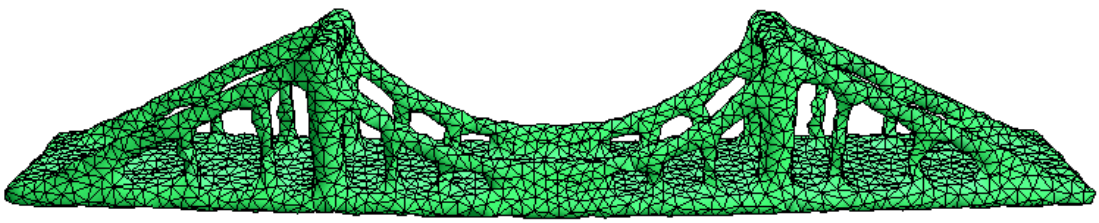


Figure 6.2: Bridge Mesh with 4000 Vertices and 12827 Elements

6.2 Interactive Cutting

In the following, we use the frog mesh as the experimental mesh in our interactive cutting simulation. The size of the global stiffness matrix for the original mesh is 2478×2478. The number of non-zero entries are 77742, therefore the fill rate of this sparse matrix is about 0.1%. We use the same setup as in the work of Jeřábková [83] and Parker *et al.* [42] in order to ensure real-time performance of our system ($N_{max} = 10$ and $\epsilon = 0.001$). It takes 9 ms for an integration step. Several cutting paths are performed and we test the

changes of performance of our system. In order to create as many cut elements as possible for testing purposes, all the cuts in our experiments completely cut through the deformable object. The final size of the stiffness matrix after cuts is recorded and it is analyzed along with the performance. We define the dimensions of the stiffness matrix as the system size of the deformable object.

The number of enriched vertices determines the size of the linear system (see Equation 4.7). In the worst case, a dissection that cuts through all elements would lead to a $O(4N^2)$ size of the stiffness matrix (i.e. a 400% increase). In practice, it is rare to see more than 60% of the vertices enriched. In fact, a complete cut may only cut through a small sets of elements, as we show with the Test 2, 3 and 4 in Table 6.3. Normally, the number of additional vertices is less than 20% in our experiments. While this number is closely related to the actual topology of the volumetric mesh, in general the number of added degrees of freedom is small. Therefore, for a complex volumetric meshes that has more elements, the number of added entries would be larger compared to a simpler mesh.

The experiments in Table 6.3 also show that the integration time after dissections scales with the number of entries in the sparse matrix. As we can obverse in Table 6.3, although the number of additional vertices is less than 20% in most scenarios, the cut still results in an additional number of non-zero entries that is larger than 20%. Therefore, it leads to an increase of the integration time that is also larger than 20%. Figure 6.3 demonstrates the comparison of the system size, the number of non-zero entries and the integration time between each test cases. All values are normalized to show the increase proportion after dissections.

We also found that the increase in the number of non-zero entries is not proportional to the number of enriched vertices. An enriched vertex is created as a cut proceeds and enriches an element that shares the vertex. We explain in Section 4.2.2 that in a single cut scenario, an vertex is only enriched once. There might be more than one enriched element that shares the same enriched vertex. Ideally, one should only add entries of the enriched elements into the stiffness matrix. However, it is worth noting that in an interactive dissection simulation, we can hardly have any prior knowledge of the cuts that might appear in the future. We could only create the topological structures for an newly enriched element in the stiffness matrix. However, if another element that shares some of its vertices is enriched later in the simulation, then the corresponding entries need to be added to the stiffness matrix. As we discuss in Section 4.2.2, the structure of the original stiffness matrix is not changed during the simulation. All additional DOFs are incrementally appended to the previous stiffness matrix, which is computationally inexpensive in the compressed sparse row format implementation of the sparse matrix. But insertion operations require re-arranging the entries of the sparse matrix, which is harmful to real-time applications. Therefore when we create an enriched vertex, we need to connect it to all directly connected vertices of the original vertex in the stiffness matrix in Vega FEM. This may result in redundant entries in the sparse matrix that are not required to model the cuts. And the additional entries of the sparse matrix would impact the performance of the system. Since we can neither predict nor constrain the cutting path driven by the user and hence we introduce the redundancy of the enriched vertex data to ensure the responsiveness of our simulation at the cost of average performance and

memory efficiency. As shown in Table 6.3, the integration time is still acceptable after non-trivial cuts. Also, the actual fill rate of the sparse matrix is still very low dispite the data redundancy, which means the efficiency of memory usage is still preserved.

Test Cases	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7
Num of Cuts	0	1	1	1	2	2	3
Integration Time	9	10	12	13	15	12	15
System Size	2478^2	2595^2	2751^2	2715^2	2901^2	2688^2	2985^2
Enriched Vertices	0	39	91	79	141	70	169
Non-Zero Entries	77742	89325	105759	102393	120789	100098	130707
Fill Rate	1.27%	1.33%	1.40%	1.39%	1.44%	1.39%	1.47%

Table 6.3: Integration Time after Dissection (msec)

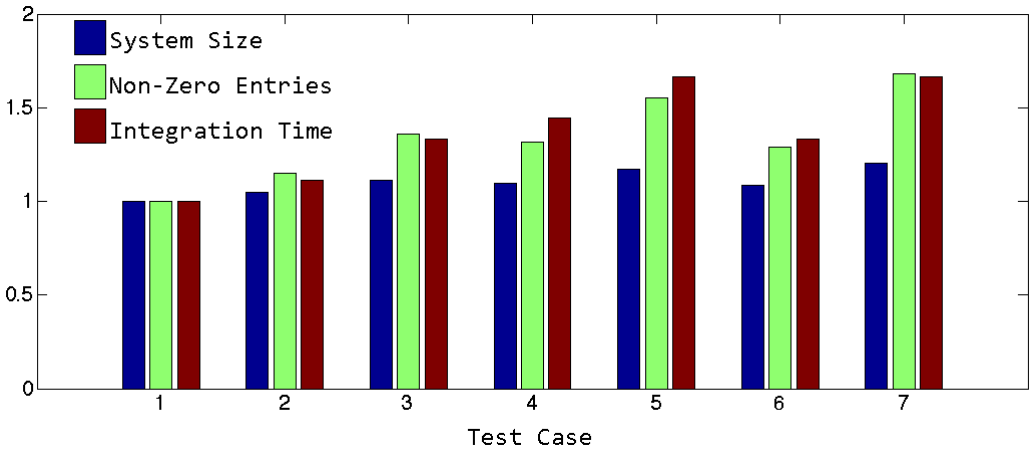


Figure 6.3: Normalized Performance Comparison after Dissections.

In our semi-progressive cutting scheme, we enrich an element as soon as it is cut completely. At each time step, our system needs to find the intersected elements using our volume collision detection method. Since our method is based on a local neighborhood search, the process is very efficient. Figure 6.4, 6.5 and 6.6 demonstrate the performance of our volume collision detection method in different scenarios. The number of newly cut elements is usually less than 10 in our scenarios. Even in the timesteps when more elements are enriched, it seldom takes more than 1.5 ms to complete, which is negligible compared to the integration time. Our method distributes the enrichments over several consecutive time steps as a cut proceeds. This distribution improves the responsiveness of our system because the user visually observes dynamic behavior of the dissected object even though our cutting scheme is not fully progressive.

It is also worth noting that the performance of our volume collision detection method is closely related to the resolution of the volumetric representation and the length of the virtual scalpel. We perform another set of experiments to collect the statistics of our volume collision detection method as shown in Table 6.4. In these experiments, the object

does not deform, and we apply a single cut that tries to cut through as many elements as possible. To minimize the influence of the length of the virtual scalpel, we set it to a large enough value so that it can always cut the deformable object completely. The results show that the resolution influences the number of newly cut elements at each step. However, the performance of our method is efficient in all scenarios tested.

Statistics	Sample Meshes			
	Turtle	Tiger	Rat	Frog
Average Time (msec)	1.5	1.4	0.9	0.5
Median Time (msec)	0.5	1.3	0.9	0.6
Average Num of Cut Elements	3.04	10.52	5.32	5.23
Median Num of Cut Elements	2	7	4	4

Table 6.4: Statistics of Volume Collision Detection

In the original implementation of the sparse matrix in the Vega FEM library, its size is determined in the pre-processing phase and then fixed during the whole simulation. However, since we incorporate the XFEM into the simulation, we need to modify the size of the linear system as a cut proceeds. We observe that the time to dynamically re-allocating the memory block during the simulation, which takes about 0.02 s to 0.7 s depending on the structure of the sparse matrix and the scenario. While the memory re-allocation only influences few time steps (normally less than 20 time steps as shown in Figure 6.4, 6.5 and 6.6) during a cut, the sudden change during the cuts creates a noticeable delay in the animation. It is very important for a real-time interactive simulation to have a less variable frame rate so that the user does not feel "lag" during the simulation.

The memory reallocation issue can be greatly alleviated by pre-allocating larger memory blocks for the sparse matrix and the vectors for the deformable object. The number of additional DOFs seldom exceeds 30% of the original number as we show in Table 6.3, therefore the vectors holding physical properties of the deformable object are pre-allocated with a size of 150% of the original DOFs. The pre-allocation for a sparse matrix is trickier because the numerical relationship does not hold for the number of non-zero entries. We observe that a 20% increase in the number of vertices results to in 50% increase in the number of non-zero entries in the stiffness matrix. In order to ensure better responsiveness of our system, we pre-allocate a sparse matrix with the size of 400% of its original size: for a sparse matrix with N rows, the memory block for each row is doubled, and N additional rows are appended with the average size (after expansion) of the previous N rows. Based on our modifications, the time of expanding the linear system can be eliminated in most scenarios. This is essentially trading in memory efficiency to the responsiveness of the system. Since the sparse matrix is already memory efficient, the cost is affordable in most scenarios. And the user can choose these two parameters based on the situation.

6.3 Test Application

We have built a test application based on our system and demonstrate several dissection scenarios on different meshes. Gravity is applied to the deformable object in all scenarios. The rendering of the cutting surface and the enriched elements in Figures 6.7 and6.8 is the work of Ling Jin. The intersected surface triangles after dissections are replaced with a set of new triangles based on the enrichment information of the intersected elements. More sophisticated re-meshing schemes of the surface could be applied, but we leave this as a future work. It is worth noting that the rendering of our system is to provide a technical view. We use neither texture nor advanced reflection in our system yet.

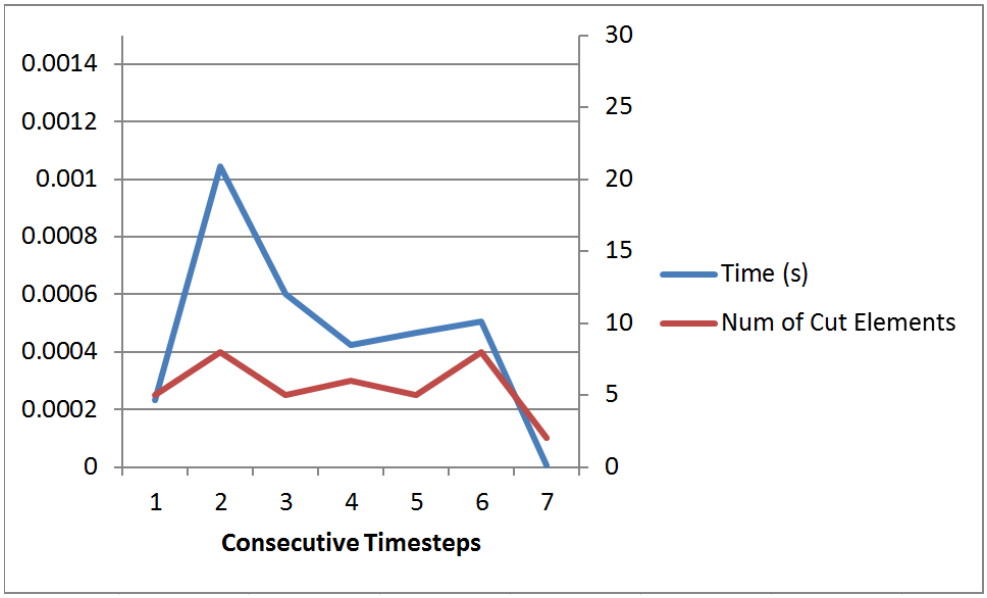
Figure 6.7 shows a complete cut scenario on the frog mesh. Figure 6.8 demonstrates that our system can also capture partial cuts. The wireframes of cut elements and the enriched vertices are rendered. Enriched vertices above a cut are marked in orange and others are in blue. We are limited to only one cut within an element, and an element can be only enriched for once. However, it is possible to create more than one cut within the deformable object as long as the cuts enrich an element not more than once. Figure 6.9 demonstrates the cutting process on a beam with two separate cuts. In Figure 6.9c and 6.9e, the cut elements are rendered to demonstrate the concept of XFEM. Each cut element are enriched with additional degrees of freedom. In Figure 6.9f, the signs of enriched vertices are rendered. Figure 6.10, 6.11 and 6.12 show more cutting patterns using our application.

6.4 Discussion

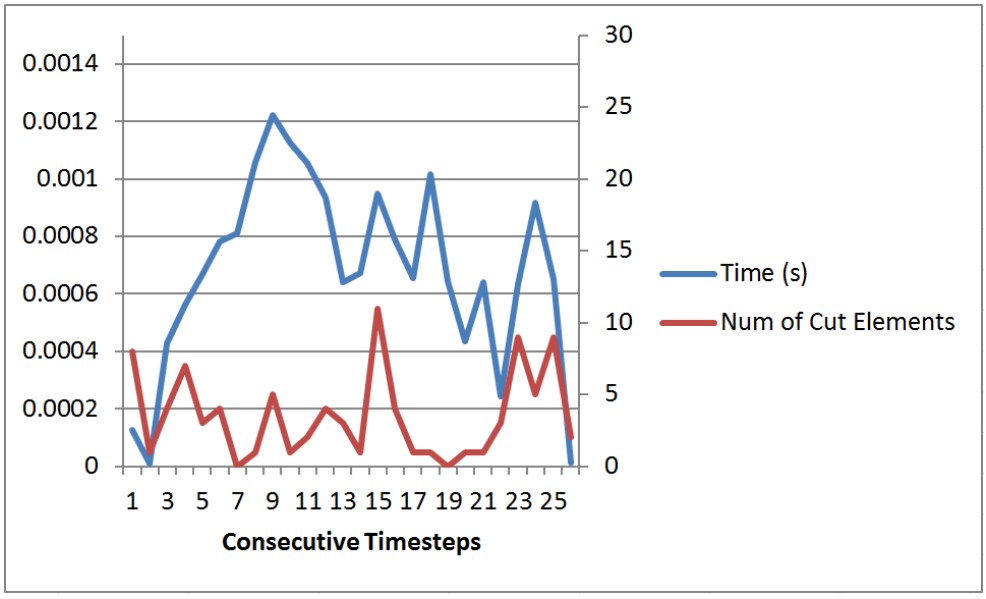
We compare the performance of our system with other recent virtual dissection methods. In terms of mesh-based methods, Niroomandi *et al.* [47] also use the XFEM to model dissections. In their work, they simulate a hexahedral mesh consisting of 8514 vertices and 7182 elements using a model reduction method. The reduced stiffness matrix has the size of 15×15, and their system is able to run at about 25 Hz. Wu *et al.* [62] build an haptic cutting simulation based on the composite FEM. The original hexahedral mesh used in their experiment consists of 173843 elements. By applying composition, they arrive at 647 composite elements with 2928 DOFs. Their simulation is running at 15 frames per second. For meshfree methods, Pauly *et al.* [9] report the performance of their method on a model with 4300 simulation nodes and 250000 surface samples. During the fracturing, the number of nodes increases to 6500 and 61000 new surface samples are added to model the cuts. Their simulation is running at an average of 22 Hz. LeBlanc *et al.* [7] build their SPH engine on top of the CUDA framework. By utilizing the parallelism of the GPU architecture, they report to able to simulate 1000 particles at 257 Hz and 60000 particles at 6.3 Hz.

In comparison, our experiment shows that the system can use a tetrahedral mesh with 12800 elements and 3321 vertices at 15 Hz. Based on the results of our experiments, we demonstrate that the performance of a FEM-based simulation is suitable for real-time interactive applications. By using the XFEM, we alleviate the performance penalty in

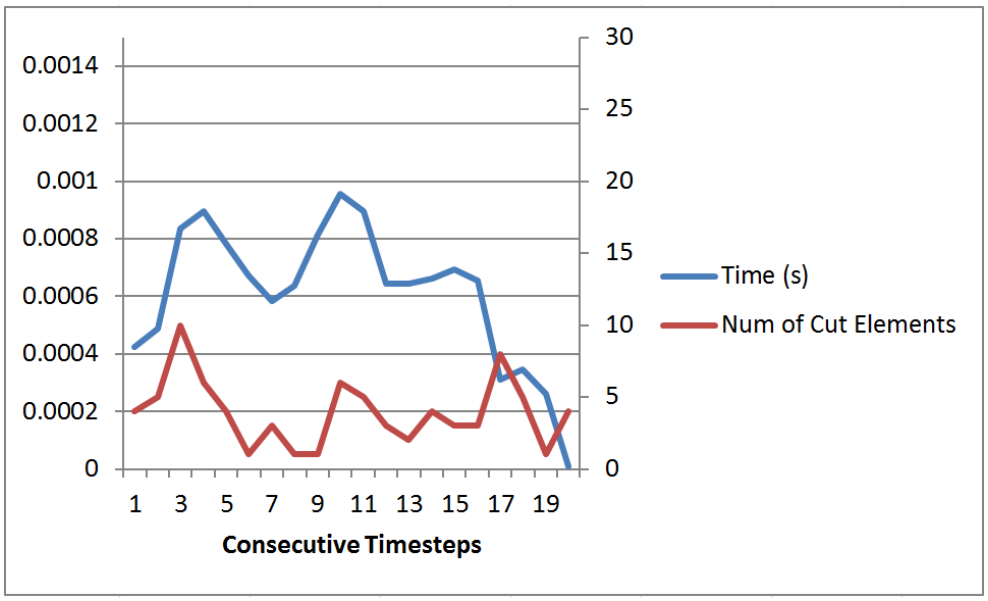
modeling dissection. Although the problem size is not quite as large as in meshfree methods such as [7], we demonstrate that it is still possible to build an interactive simulation with non-trivial models on a normal PC. We also note that it is difficult to compare the number of particles in meshfree methods with the number of vertices in FEM methods. We find that using XFEM is helpful to control the increase of the DOFs to model cuts. In terms of re-meshing methods, the work of Mor and Kanade [12] creates 5.2 new elements on average to replace an intersected element. The meshfree method of Pauly *et al.* [9] also leads to a 40% increase in the number of simulation nodes. In comparison, the added number of DOFs using XFEM seldom exceeds 20% of the original DOFs. Since our simulation is directly with the full volumetric mesh, there is still potential to improve the performance by employing model reduction. Also, since the dynamics of our system is essentially computed using a single core, there is still potential to improve the performance by utilizing parallelism, e.g., available on multi-core architectures and graphics processing units, which are becoming prevalent on consumer PC. Sin *et al.* [20] report that their parallel version of Vega gains significant speed-up by going to two and three cores from a single core.



(a) Test Case 2

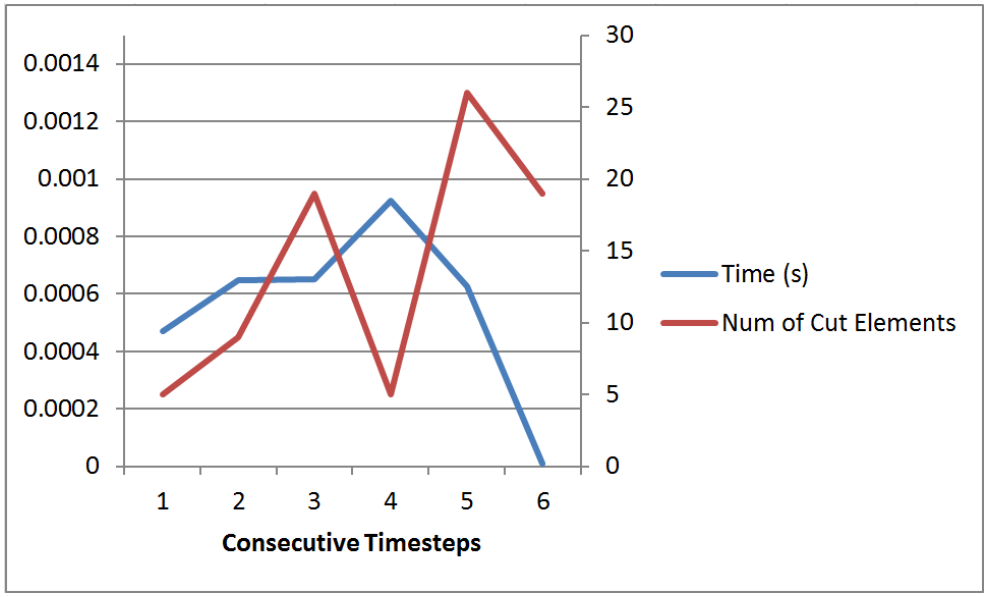


(b) Test Case 3

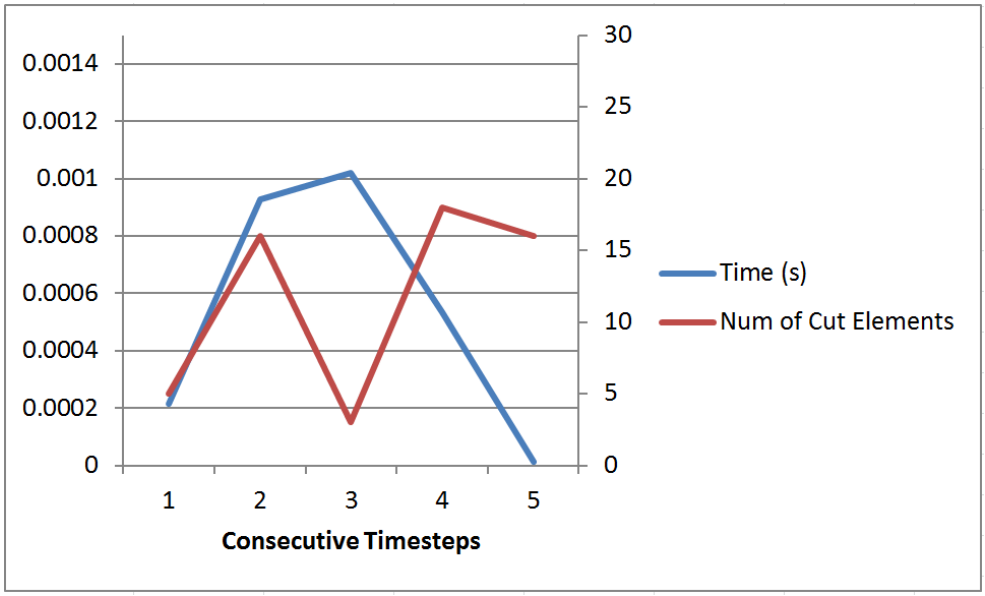


(c) Test Case 4

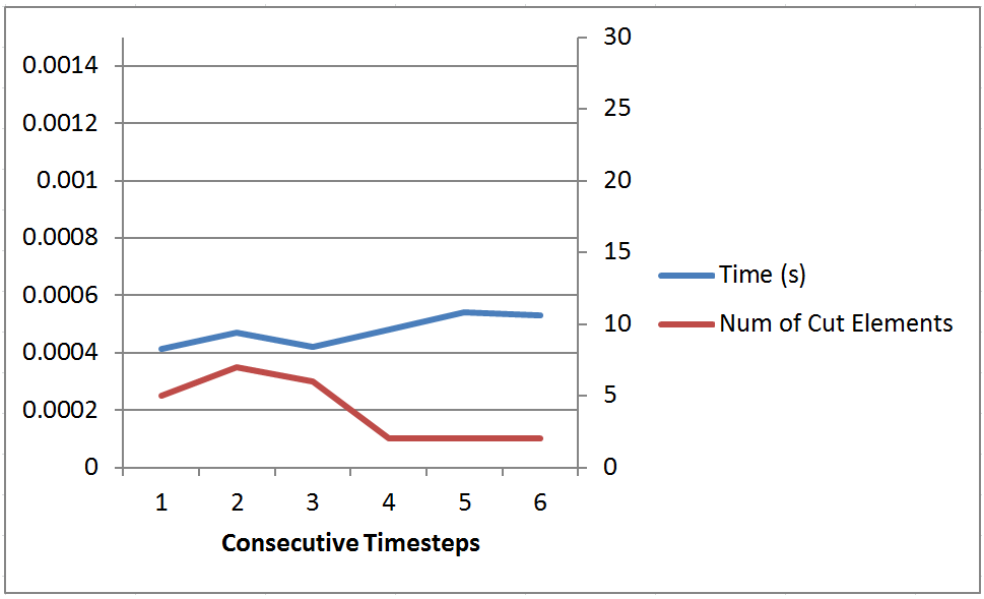
Figure 6.4: Volume Collision Detecion with a Single Cut



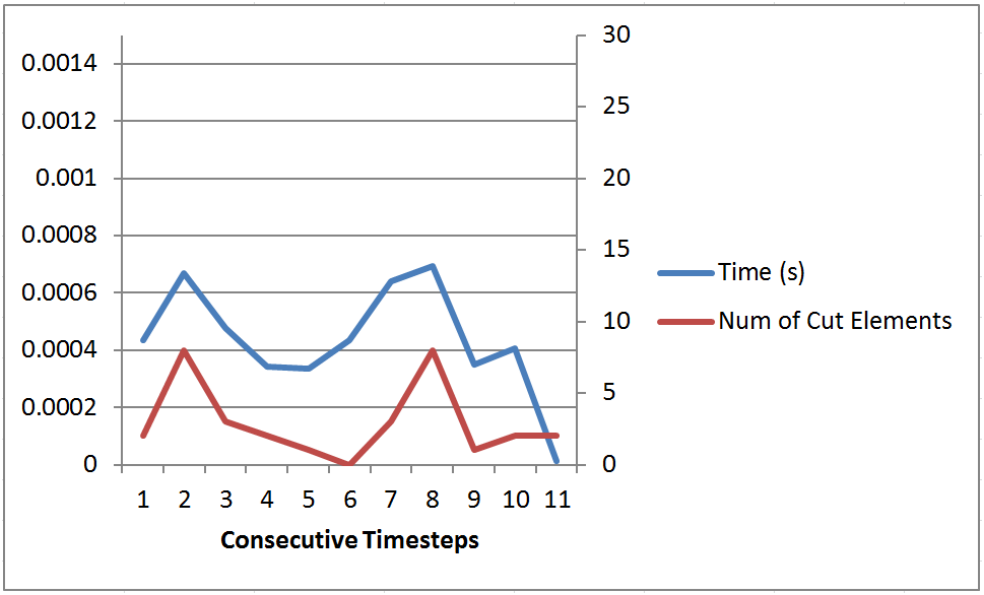
(a) The First Cut in Test Case 5



(b) The Second Cut in Test Case 5

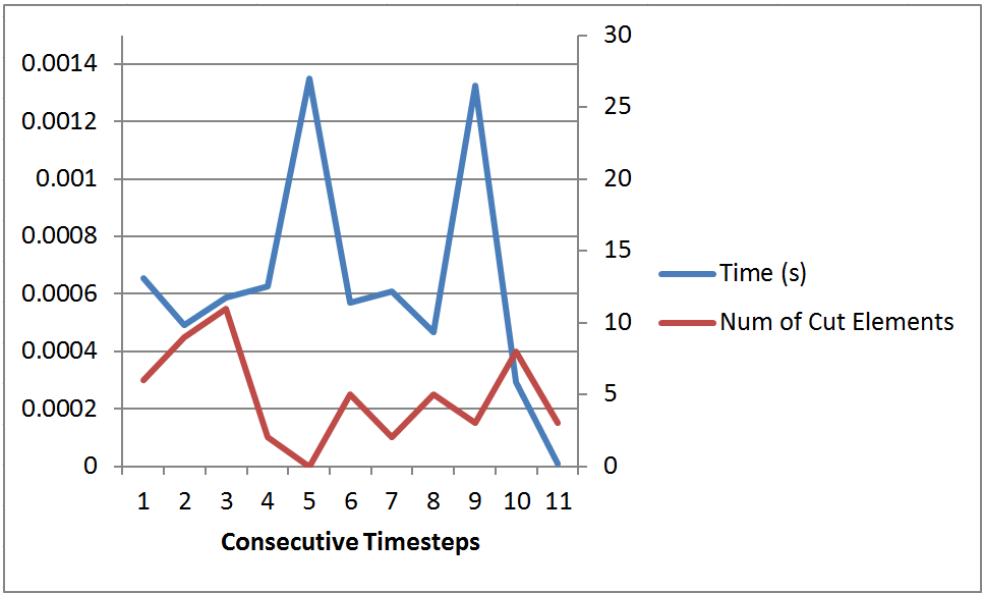


(c) The First Cut in Test Case 6

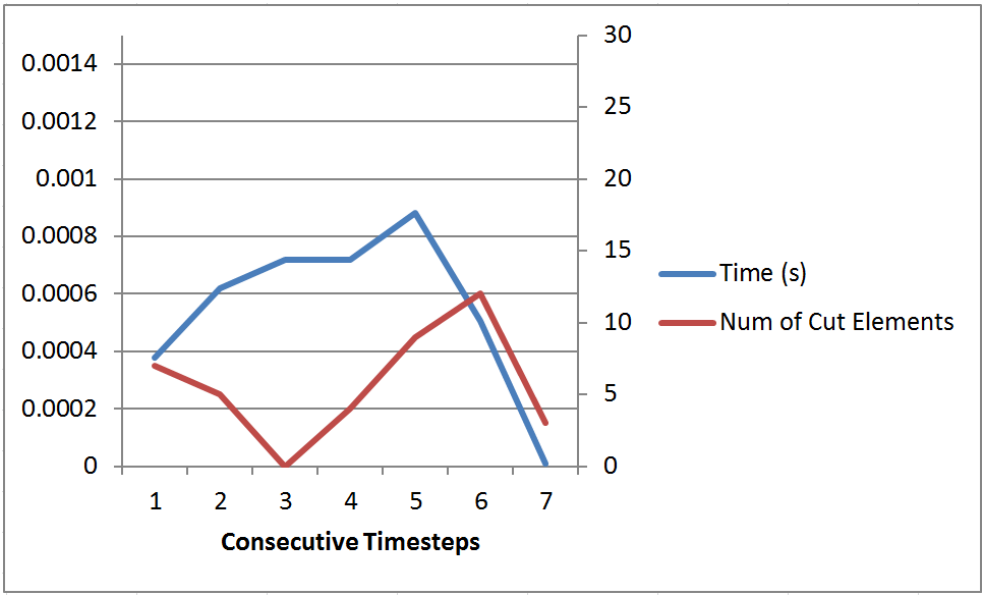


(d) The Second Cut in Test Case 6

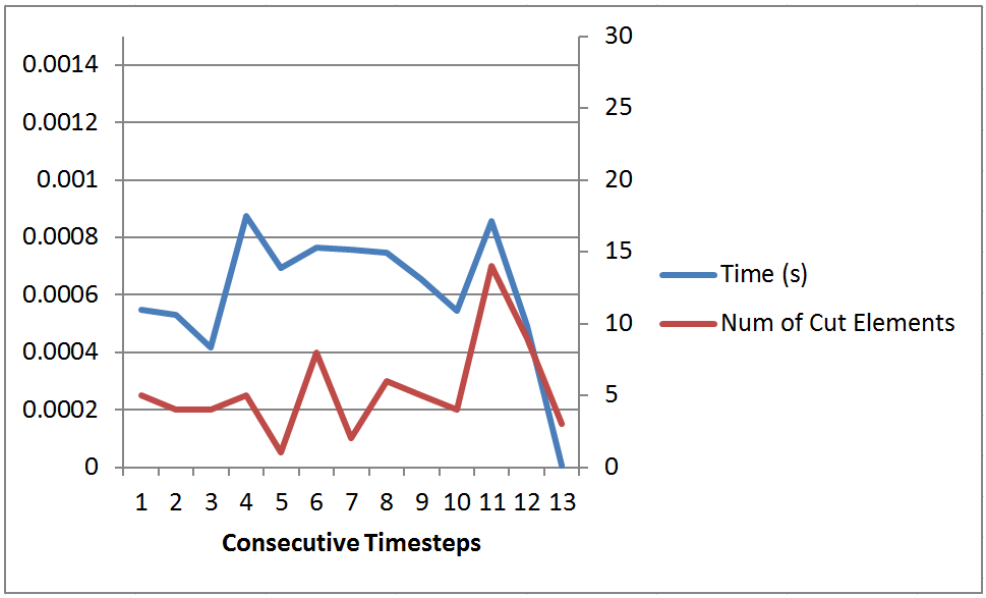
Figure 6.5: Volume Collision Detecion in Two Separate Cuts



(a) The First Cut in Test Case 7



(b) The Second Cut in Test Case 7



(c) The Third Cut in Test Case 7

Figure 6.6: Volume Collision Detecion in Three Separate Cuts

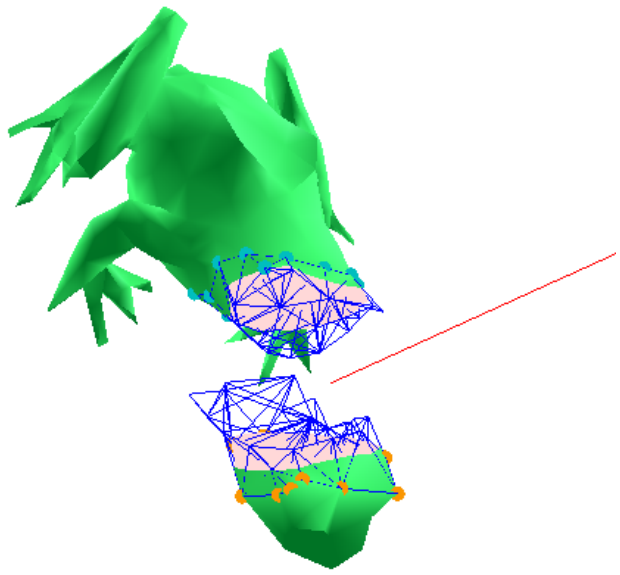


Figure 6.7: Completely Cut Frog

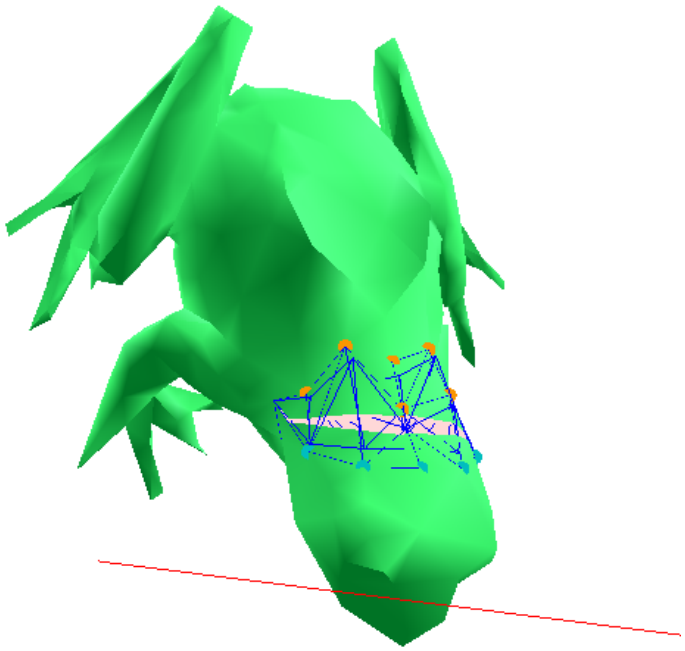
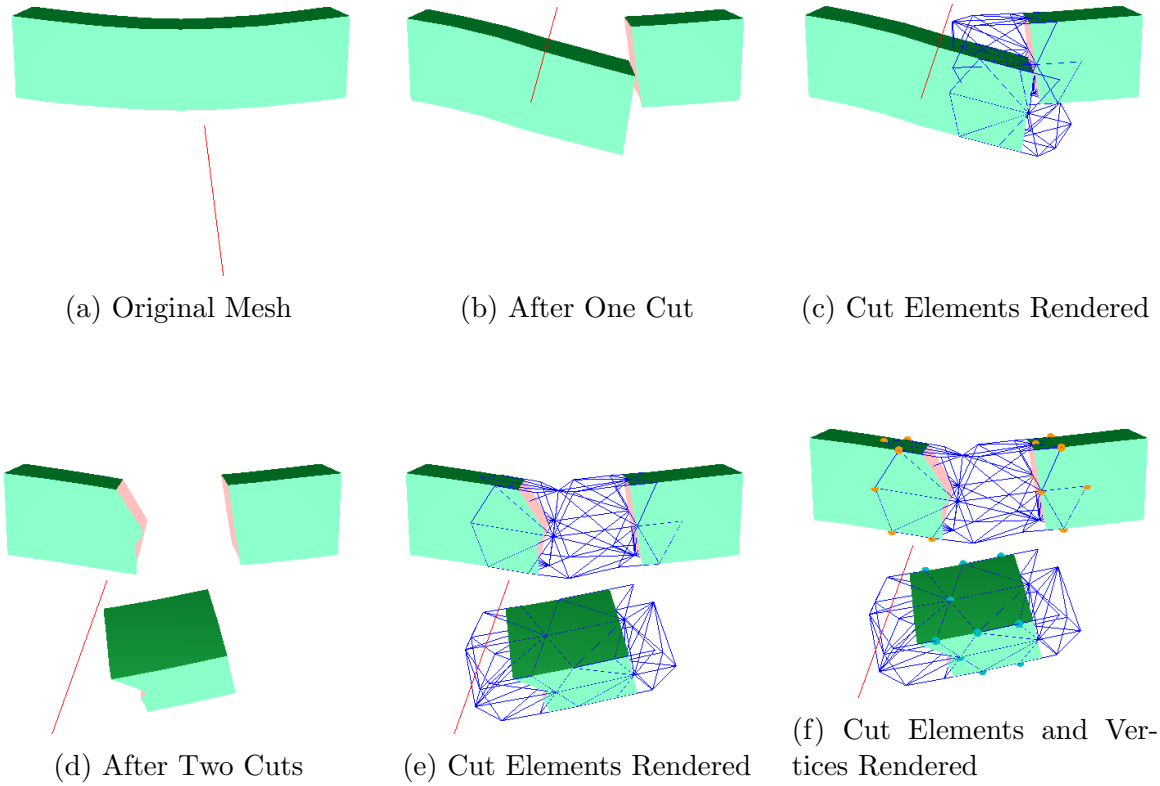


Figure 6.8: Partially Cut Frog



(a) Original Mesh (b) After One Cut (c) Cut Elements Rendered

(d) After Two Cuts (e) Cut Elements Rendered (f) Cut Elements and Vertices Rendered

Figure 6.9: Semi-Progressive Cut on a Beam

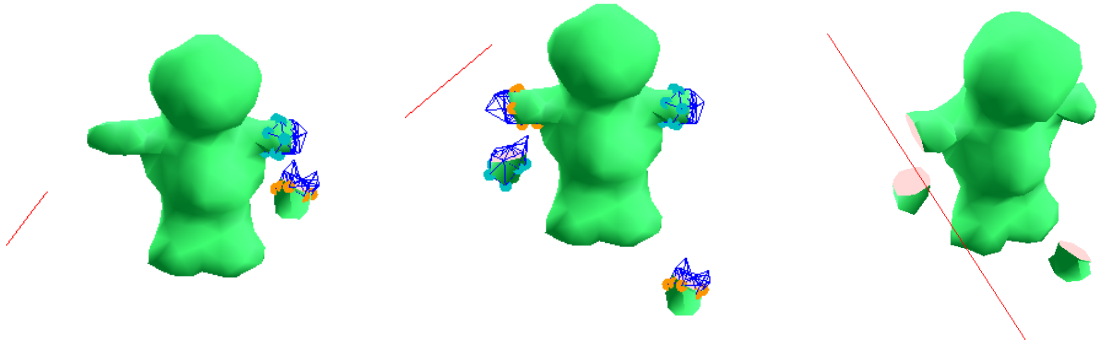


Figure 6.10: Semi-Progressive Cut on a Turtle

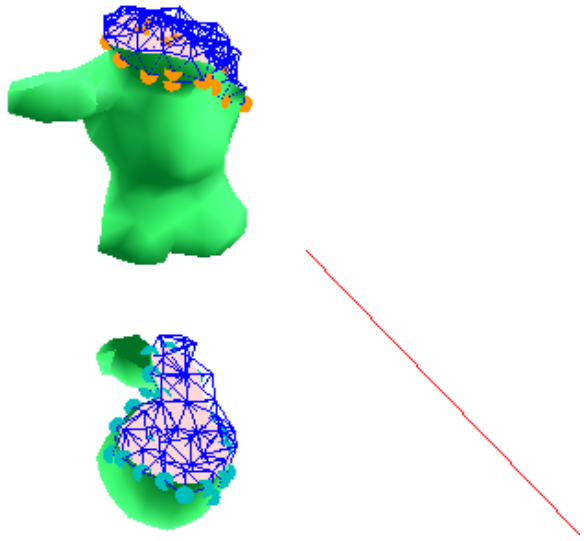


Figure 6.11: Another Cut Pattern on a Turtle

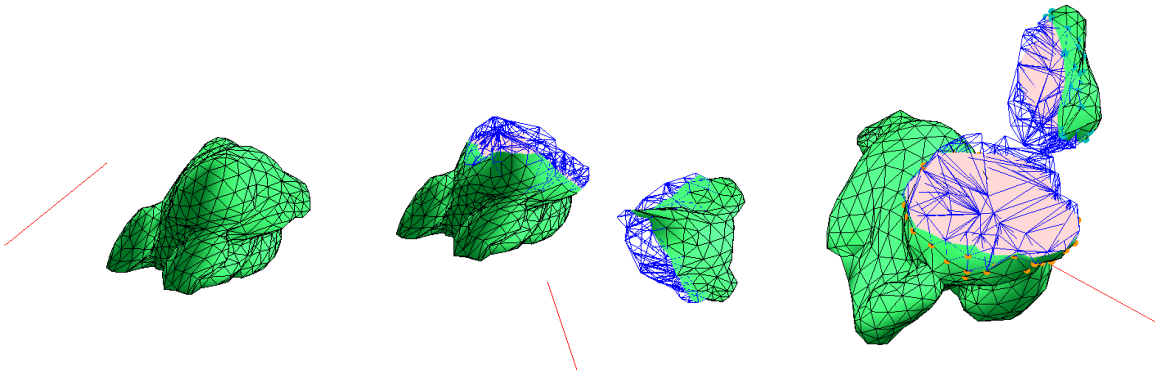


Figure 6.12: Semi-Progressive Cut on a Tiger with a Horizontal Force Field

Chapter 7

Conclusions

In this thesis, we present a stable and efficient virtual dissection simulation based on XFEM. We outline the design of our system which couples the XFEM-based dynamic simulation and the haptic interaction. We propose semi-progressive virtual cutting based on an improved volume collision detection method, which can efficiently detect the intersected elements within the tetrahedral mesh against a splitting plane. The system is built on top of open source software packages that are freely available. The performance of our system is compared with the original implementation of Vega FEM and other virtual dissection methods. We found that it is possible to build realistic dissection simulations on low-end PCs which do not require costly configuration and equipment.

The modeling of the dissected deformable object is based on the XFEM framework by Jeřábková and Kuhlen [17]. Our system based on the XFEM is efficient, because the topology of the volumetric mesh does not need to be changed even when there are dissections within the mesh. This avoids the unnecessary computational overhead of changing the topology in real-time. Since our project aims to provide an economic solution for dissection simulations, the efficiency allows our system to provide more visually plausible results with a limited computational power. Our experiments show that the dissections using the XFEM only create a smaller number of additional DOFs compared to [9, 12].

A haptic device is used as the interface between the user and our system. By using the haptic device, the user are given enough explore the virtual 3-D environment . It is necessary to pursue stability of the system since the user input is unpredictable in such scenarios. For virtual dissection methods that subdivides the intersected elements [54, 55, 12, 58], slivers or ill-shaped elements created by the dissections would hurt the stability of the simulation. There is no practical way to prevent the user from creating slivers or ill-shaped elements during the dissection. The fact that the volumetric mesh remains unchanged using the XFEM that there are no slivers or ill-conditioned elements created by cuts, therefore the stability of the simulation based on the XFEM is improved [17].

To provide better immersion to the user, our system adapts a semi-progressive cutting method to the simulate dissection procedure. This method only deals with complete cuts of elements (i.e. an element is cut into two separate parts) and ignore any partial cuts of the elements during the simulation. It is not as computationally expensive as a fully

progressive cutting simulation, while it provides enough responsiveness to the user because he can observe the deformation of the dissected object during the cutting interaction. The semi-progressive cutting scheme is based on a volume collision detection method derived from the work of Bielser and Gross [55]. The original method is designed for a fully progressive cutting procedure, which means the underlying topology of the volumetric mesh is changed after each cut. However, we need to preserve the original topology of the volumetric mesh in the XFEM framework. Therefore, we propose a modified version of the volume collision detection method, which is general to XFEM-based simulation and more appropriate with haptic interactions. The performance of the volume detection method is tested in Section 6.2, which shows the feasibility of our method in performance-critical applications.

We present the design of our system based on the XFEM framework using several open-source libraries. We outline the roles of each library in our system, and present a general design of a dissection simulation with haptic supports. Based on our design, it is easy to change the underlying implementation details to fit different user's needs. Our system is able to deliver real-time performance on a reasonable complex tetrahedral mesh. We provide detailed performance information on several tetrahedral meshes that have different resolutions. It is shown that the cutting procedure does not cause a huge computational overhead on the performance. A complete cut may only cut through a small set of elements (normally less than 20% in our experiments), which means the linear system does not grow dramatically in most cases. We find that the time spent on the linear system expansion can be alleviated by memory pre-allocation. We implement a test application to demonstrate a useful dissection simulation for low-end computational environments.

7.1 Future Work

One limitation of our system is that there is only one cut within any enriched element in the tetrahedral mesh. As mentioned by Jeřábková and Kuhlen [17], it is possible to supports multiple cuts within an element by defining a higher level discontinuous function. Also, in the XFEM framework, the discontinuous function is global and all enriched vertices are influenced by it. In theory, it is possible to define an arbitrary discontinuous function with a complex splitting surface. However, this may cause many issues even just with a single cut. For example, if the cutting path is spiral as in Figure 7.1, the sign of the vertex circled in red is ambiguous in our current XFEM framework. This issue may be solved by finding a more appropriate discontinuous function instead of the shifted enrichment function we are using.

Although we use a haptic device as the user interface to interactively change the dynamic states of the deformable object in the scene, we do not have a haptic rendering method to provide force feedback to the user during cutting. And as we mention earlier, the haptic rendering for dissections is still an open research problem. The measurement-based model is computationally inexpensive method to model the force-deformation relationship without much priori knowledge of the object. Cretu [84] proposes a framework to model

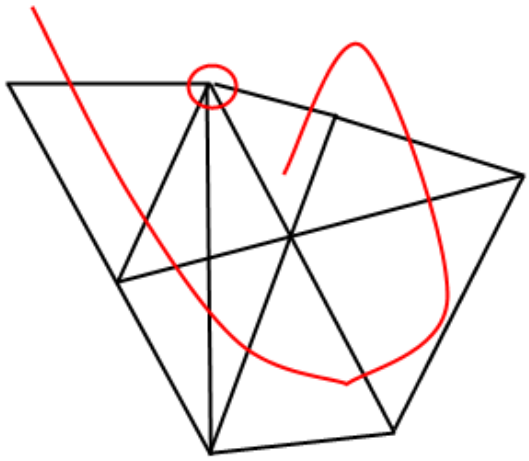


Figure 7.1: Spiral Cut Path

deformable objects by employing neural networks on experimental data acquired from measurement. The deformation behavior of an object is modeled by a neural network which captures a set of features based on the collected data. Zhou [85] presents an estimation framework based on an adaptive Kalman filter to acquire precise velocity and acceleration for haptic interfaces which may only have limited sensor resolution. He also studies that the effective mass and viscous damping of a haptic interface may cause the non-uniform hardness perception to the user. He proposes three different methods to compensate this perception and presents the subjective evaluations of them. Lang and Andrew propose the haptic rendering for textures using the measurement-based model of contact forces [86] [87], and they also presents a 3D scanning pipeline for the acquisition of visual texture [88]. Using the pre-measured samples of contact force to provide the haptic force in the cutting scenario may be a feasible solution.

There is still room for improving the performance of our system. Our system is implemented only using single-core. Both Jeřábková [83] and Sin *el at.* [20] obverse an improvement of efficiency by using a multi-threaded implementation. LeBlanc *et al.* [7] demonstrates an improvement of efficiency by utilizing the computational power of the GPU. Our system can definitely benefit from high performance implementations of the sparse matrix, the PCG solver and the force model by using either multi-threading or GPUs. Since the mult-core CPUs and GPUs are prevalent nowadays, it is reasonable to adapt a parallel solution to make use of this additional computational power. Although our system is able to process a reasonable complex tetrahedral mesh, the finer resolution of the mesh is always preferred in order to provide better immersion to the user.

It is possible to decouple the surface mesh from the volumetric mesh, and therefore uses a finer surface mesh for rendering while use a coarse volumetric mesh for simulation. However, it is still not clear how to re-mesh the surface mesh for the better visualization of the dissections while still preserving the topology of the underlying volumetric mesh. Also, our semi-progressive cutting scheme models the cut within an element as a plane. Re-meshing the surface based on this plane is not able to capture the detailed cutting path on the deformable object. A more sophisticated method is required to visualize the dissections in details while still to keep the simplicity of the dissection modeling using the XFEM.

References

[1] H. R. Malone, O. N. Syed, M. S. Downes, A. L. D'Ambrosio, D. O. Quest, and M. G. Kaiser, "Simulation in neurosurgery: a review of computer-based simulation environments and their surgical applications," *Neurosurgery*, vol. 67, no. 4, pp. 1105–1116, 2010.

[2] A. G. Gallagher, N. E. Seymour, J.-A. Jordan-Black, B. P. Bunting, K. McGlade, and R. M. Satava, "Prospective, randomized assessment of transfer of training (ToT) and transfer effectiveness ratio (TER) of virtual reality simulation training for laparoscopic skill acquisition," *Annals of surgery*, vol. 257, no. 6, pp. 1025–1031, 2013.

[3] N. Iwata, M. Fujiwara, Y. Kodera, C. Tanaka, N. Ohashi, G. Nakayama, M. Koike, and A. Nakao, "Construct validity of the LapVR virtual-reality surgical simulator," *Surgical endoscopy*, vol. 25, no. 2, pp. 423–428, 2011.

[4] F. Conti, F. Barbagli, D. Morris, and C. Sewell, "Chai 3d." Available: `http://www.chai3d.org/`. [Online].

[5] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH fluids in computer graphics," in *Eurographics 2014-State of the Art Reports*, pp. 21–42, The Eurographics Association, 2014.

[6] M. Becker, M. Ihmsen, and M. Teschner, "Corotated SPH for deformable solids," in *Proceedings of the Fifth Eurographics conference on Natural Phenomena*, pp. 27–34, Eurographics Association, 2009.

[7] S. LeBlanc, P. Boyer, and C. Joslin, "Modelling and animation of impact and damage with smoothed particle hydrodynamics," *The Visual Computer*, vol. 30, no. 6-8, pp. 909–917, 2014.

[8] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 471–478, ACM, 2005.

[9] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, and L. J. Guibas, "Meshless animation of fracturing solids," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 957–964, ACM, 2005.

[10] D. Steinemann, M. A. Otaduy, and M. Gross, "Fast arbitrary splitting of deforming objects," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 63–72, Eurographics Association, 2006.

[11] N. Pietroni, F. Ganovelli, P. Cignoni, and R. Scopigno, "Splitting cubes: a fast and robust technique for virtual cutting," *The Visual Computer*, vol. 25, no. 3, pp. 227–239, 2009.

[12] A. B. Mor and T. Kanade, "Modifying soft tissue models: Progressive cutting with minimal new element creation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2000*, pp. 598–607, Springer, 2000.

[13] H.-W. Nienhuys and A. F. van der Stappen, "Supporting cuts and finite element deformation in interactive surgery simulation," in *Procs. of the Fourth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI01)*, pp. 145–152, 2001.

[14] N. Molino, Z. Bao, and R. Fedkiw, "A virtual node algorithm for changing mesh topology during simulation," *ACM Transactions on Graphics*, vol. 23, pp. 385–392, Aug. 2004.

[15] E. Sifakis, K. G. Der, and R. Fedkiw, "Arbitrary cutting of deformable tetrahedralized objects," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pp. 73–80, 2007.

[16] P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross, "Enrichment textures for detailed cutting of shells," *ACM Transactions on Graphics*, vol. 28, pp. 50:1–50:10, July 2009.

[17] L. Jerábková and T. Kuhlen, "Stable cutting of deformable objects in virtual environments using XFEM," *Computer Graphics and Applications, IEEE*, vol. 29, pp. 61 –71, march-april 2009.

[18] M. Kremer, D. Bommes, and L. Kobbelt, "OpenVolumeMesh - a versatile index-based data structure for 3D polytopal complexes," in *Proceedings of the 21st International Meshing Roundtable*, pp. 531–548, Springer, 2013.

[19] E. Coumans, "Bullet physics library." Available: `http://bulletphysics.org`. [Online].

[20] F. S. Sin, D. Schroeder, and J. Barbič, "Vega: Non-linear FEM deformable object simulator," in *Computer Graphics Forum*, vol. 32, pp. 36–48, 2013.

[21] D. E. Breen, D. H. House, and M. J. Wozny, "Predicting the drape of woven cloth using interacting particles," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, (New York, NY, USA), pp. 365–372, ACM, 1994.

[22] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behaviour," in *Proceedings of Graphics Interface*, 1995.

[23] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 43–54, 1998.

[24] H. N. Ng and R. L. Grimsdale, "Computer graphics techniques for modeling cloth," *Computer Graphics and Applications, IEEE*, vol. 16, pp. 28–41, Sep 1996.

[25] N. Magnenat-Thalmann, F. Cordier, M. Keckeisen, S. Kimmerle, R. Klein, and J. Meseth, "Simulation of clothes for real-time applications," *Eurographics 2004, Tutorials 1: Simulation of Clothes for Real-time Applications*, vol. 2, 2004.

[26] M. Desbrun, P. Schröder, and A. Barr, "Interactive animation of structured deformable objects," in *Proceedings of Graphics Interface*, pp. 1–8, 1999.

[27] M. Müller and N. Chentanez, "Solid simulation with oriented particles," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 92, ACM, 2011.

[28] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," in *Vriphys: 3rd Workshop in Virtual Realitiy, Interactions, and Physical Simulation*, pp. 71–80, The Eurographics Association, 2006.

[29] S. F. Gibson, "3D Chainmail: A fast algorithm for deforming volumetric objects," in *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, pp. 149–ff., 1997.

[30] F. Conti, O. Khatib, and C. Baur, "Interactive rendering of deformable objects based on a filling sphere modeling approach," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, pp. 3716–3721, Sept 2003.

[31] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," in *Computer Graphics Forum*, vol. 25, pp. 809–836, 2006.

[32] U. Meier, O. López, C. Monserrat, M. C. Juan, and M. Alcaniz, "Real-time deformable models for surgery simulation: a survey," *Computer methods and programs in biomedicine*, vol. 77, no. 3, pp. 183–197, 2005.

[33] J. Wu, R. Westermann, and C. Dick, "Physically-based simulation of cuts in deformable bodies: A survey," in *Eurographics State-of-the-Art Reports*, pp. 1–19, The Eurographics Association, 2014.

[34] S. Cotin, H. Delingette, and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, pp. 62–73, Jan 1999.

[35] D. L. James and D. K. Pai, "ArtDefo: Accurate real time deformable objects," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pp. 65–72, 1999.

[36] Y. Zhuang and J. Canny, "Real-time global deformations," in *Proc. 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.

[37] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr, "Dynamic real-time deformations using space & time adaptive sampling," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pp. 31–36, 2001.

[38] X. Wu, M. S. Downes, T. Goktekin, and F. Tendick, "Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes," *Computer Graphics Forum*, vol. 20, no. 3, pp. 349–358, 2001.

[39] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, pp. 131–140, 2004.

[40] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pp. 49–54, 2002.

[41] M. Müller and M. Gross, "Interactive virtual materials," in *Proceedings of Graphics Interface*, pp. 239–246, 2004.

[42] E. G. Parker and J. F. O'Brien, "Real-time deformation and fracture in a game environment," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pp. 165–175, 2009.

[43] M. Bro-Nielsen and S. Cotin, "Real-time volumetric deformable models for surgery simulation using finite elements and condensation," *Computer Graphics Forum*, vol. 15, no. 3, pp. 57–66, 1996.

[44] J. Barbič and D. L. James, "Real-time subspace integration for St. Venant-Kirchhoff deformable models," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 982–990, ACM, 2005.

[45] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum, "Subspace gradient domain mesh deformation," in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 1126–1134, ACM, 2006.

[46] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, "Preserving topology and elasticity for embedded deformable models," in *ACM Transactions on Graphics (TOG)*, vol. 28, p. 52, ACM, 2009.

[47] S. Niroomandi, I. Alfaro, D. Gonzalez, E. Cueto, and F. Chinesta, "Real-time simulation of surgery by reduced-order modeling and X-FEM techniques," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 28, no. 5, pp. 574–588, 2012.

[48] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly notices of the royal astronomical society*, vol. 181, no. 3, pp. 375–389, 1977.

[49] J. Stam and E. Fiume, "Depicting fire and other gaseous phenomena using diffusion processes," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 129–136, ACM, 1995.

[50] M. Desbrun, M.-P. Cani, *et al.*, "Smoothed particles: A new paradigm for animating highly deformable bodies," in *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pp. 61–76, 1996.

[51] B. Solenthaler, J. Schläfli, and R. Pajarola, "A unified particle model for fluid–solid interactions: Research articles," *Computer Animation and Virtual Worlds*, vol. 18, no. 1, pp. 69–82, 2007.

[52] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point based animation of elastic, plastic and melting objects," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 141–151, Eurographics Association, 2004.

[53] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutré, and M. Gross, "A unified lagrangian approach to solid-fluid animation," in *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, pp. 125–148, IEEE, 2005.

[54] D. Bielser, V. A. Maiwald, and M. H. Gross, "Interactive cuts through 3-dimensional soft tissue," *Computer Graphics Forum*, vol. 18, no. 3, pp. 31–38, 1999.

[55] D. Bielser and M. H. Gross, "Interactive simulation of surgical cuts," in *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on*, pp. 116–442, IEEE, 2000.

[56] H. Zhang, S. Payandeh, and J. Dill, "On cutting and dissection of virtual deformable objects," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 4, pp. 3908–3913, April 2004.

[57] S. Cotin, H. Delingette, and N. Ayache, "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," *The Visual Computer*, vol. 16, no. 8, pp. 437–452, 2000.

[58] H.-W. Nienhuys and A. F. van der Stappen, "Combining finite element deformation with cutting for surgery simulations," in *Eurographics short-presentations*, pp. 143–152, 2000.

[59] L. Jeřábková, G. Bousquet, S. Barbier, F. Faure, and J. Allard, "Volumetric modeling and interactive cutting of deformable bodies," *Progress in biophysics and molecular biology*, vol. 103, no. 2, pp. 217–224, 2010.

[60] M. Seiler, D. Steinemann, J. Spillmann, and M. Harders, "Robust interactive cutting based on an adaptive octree simulation mesh," *The Visual Computer*, vol. 27, no. 6-8, pp. 519–529, 2011.

[61] C. Dick, J. Georgii, and R. Westermann, "A hexahedral multigrid approach for simulating cuts in deformable objects," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 11, pp. 1663–1675, 2011.

[62] J. Wu, R. Westermann, and C. Dick, "Real-time haptic cutting of high resolution soft tissues," *Studies in Health Technology and Informatics (Proc. Medicine Meets Virtual Reality 2014)*, vol. 196, pp. 469–475, 2014. Published by IOS Press.

[63] J. Wu, C. Dick, and R. Westermann, "Efficient collision detection for composite finite element simulation of cuts in deformable bodies," *The Visual Computer (Proc. CGI 2013)*, vol. 29, no. 6-8, pp. 739–749, 2013.

[64] B. E. Insko, *Passive haptics significantly enhances virtual environments*. PhD thesis, University of North Carolina, 2001.

[65] D. Morris, H. Z. Tan, F. Barbagli, T. Chang, and K. Salisbury, "Haptic feedback enhances force skill learning.," in *WHC*, vol. 7, pp. 21–26, 2007.

[66] M. C. Lin and M. A. Otaduy, "Recent advances in haptic rendering & applications," in *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Courses: Los Angeles, California*, vol. 2005, 2005.

[67] S. Misra, K. Ramesh, and A. M. Okamura, "Modeling of tool-tissue interactions for computer-based surgical simulation: a literature review," *Presence: Teleoperators and Virtual Environments*, vol. 17, no. 5, pp. 463–491, 2008.

[68] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, *et al.*, "Collision detection for deformable objects," *Computer Graphics Forum*, vol. 24, no. 1, pp. 61–81, 2005.

[69] M. A. Otaduy and M. C. Lin, "Introduction to haptic rendering," in *ACM SIGGRAPH 2005 Courses*, p. 3, ACM, 2005.

[70] C. B. Zilles and J. K. Salisbury, "A constraint-based god-object method for haptic display," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 3, pp. 146–151, 1995.

[71] D. C. Ruspini, K. Kolarov, and O. Khatib, "The haptic display of complex graphical environments," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 345–352, ACM Press/Addison-Wesley Publishing Co., 1997.

[72] Y. Zhuang and J. Canny, "Haptic interaction with global deformations," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 3, pp. 2428–2433, 2000.

[73] M. Mahvash and V. Hayward, "Haptic rendering of cutting: A fracture mechanics approach," *Haptics-e*, vol. 2, no. 3, pp. 1–12, 2001.

[74] A. M. Okamura, R. J. Webster III, J. T. Nolin, K. Johnson, and H. Jafry, "The haptic scissors: cutting in virtual environments," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 1, pp. 828 – 833 vol.1, sept. 2003.

[75] M. Mahvash, L. M. Voo, D. Kim, K. Jeung, J. Wainer, and A. M. Okamura, "Modeling the forces of cutting with scissors," *Biomedical Engineering, IEEE Transactions on*, vol. 55, pp. 848–856, March 2008.

[76] A. Witkin and D. Baraff, "Physically-based modeling: Principles and practice," *SIGGRAPH 2001 Course Notes*, p. 28, 2001.

[77] M. Müller, J. Stam, D. James, and N. Thürey, "Real time physics: class notes," in *ACM SIGGRAPH 2008 classes*, p. 88, 2008.

[78] E. Sifakis and J. Barbic, "FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction," in *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, pp. 20:1–20:50, 2012.

[79] H. Si, "TetView: A tetrahedral mesh and piecewise linear complex viewer." Available: `http://wias-berlin.de/software/tetgen/tetview.html`. [Online].

[80] C. Ericson, *Real-time collision detection*. Taylor & Francis US, 2005.

[81] G. v. d. Bergen, "Efficient collision detection of complex deformable models using AABB trees," *Journal of Graphics Tools*, vol. 2, no. 4, pp. 1–13, 1997.

[82] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt, "OpenMesh - a generic and efficient polygon mesh data structure." Available: `http://www.openmesh.org/`. [Online].

[83] L. Jerábková, *Interactive cutting of finite elements based deformable objects in virtual environments*. PhD thesis, Universitätsbibliothek, 2007.

[84] A.-M. Cretu, *Experimental data acquisition and modeling of three-dimensional deformable objects using neural networks*. PhD thesis, University of Ottawa, 2009.

[85] J. Zhou, *Velocity and Acceleration Estimation and Uniform Hardness Perception in 6-DOF Haptic Rendering*. PhD thesis, University of Ottawa, 2010.

[86] S. Andrews and J. Lang, "Haptic texturing based on real-world samples," in *Haptic, Audio and Visual Environments and Games, 2007. HAVE 2007. IEEE International Workshop on*, pp. 142 –147, oct. 2007.

[87] J. Lang and S. Andrews, "Measurement-based modeling of contact forces and textures for haptic rendering," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 3, pp. 380–391, 2011.

[88] S. Andrews and J. Lang, "Interactive scanning of haptic textures and surface compliance," in *3-D Digital Imaging and Modeling, 2007. 3DIM '07. Sixth International Conference on*, pp. 99–106, Aug 2007.