School of Electrical Engineer and Computing
Department of Computing

Real-Time Hybrid Cutting with Dynamic Fluid Visualization for Virtual Surgery

Jie Peng

This thesis is presented for the degree of

Doctor of Philosophy
of
Curtin University

November 2013

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: Jie Peng

Date: 8th November 2013

Abstract

Virtual simulation techniques have experienced a great deal of progress in the last couple of decades. It has been anticipated that such techniques could be very beneficial for medical training and pre-surgery planning. It is widely accepted that a reform in medical teaching must be made to meet today's high volume training requirements. Receiving pre-training in a core set of surgical skills and procedures before novice practitioners are exposed to the traditional apprenticeship training model could reduce both the skill acquisition time and the risks patients are exposed to due to surgeon inexperience. Virtual simulation offers a potential method of providing such trainings and some current medical training simulations integrate haptic and visual feedback to enhance procedure learning. One of the most important advantages of computer simulators for surgical training is the opportunity they offer for independent learning, so that the novice doctors could learn and practice surgical skills as many times as required which is not possible in traditional apprenticeship training model. Even for experienced surgeons, virtual simulations provide them the opportunity to sharpen their skills on new surgical procedures or to plan and practice prior to performing an actual surgery.

The purpose of this project is to explore the capability of Virtual Reality (VR) technology to develop a training simulator for surgical cutting and bleeding in a general surgery. Two specific research contributions are presented.

Firstly, this thesis shows that surface-only cutting is not sufficient for resembling the real surgical cutting as a great deal of information about interior structures would be generally disregarded. Although volumetric model could present large information about the internal structures and material properties, physically based realistic topology modification and deformation on volume data is far from being able to implement in real time, even if GPU accelerated integration schemes are used. This thesis proposes to combine the surface model and volume model into a hybrid model for surgical cutting and deformation. The outer surface is responsible for smooth realistic cutting while the inner volumetric model is employed for groove generation and presenting internal structures after cutting. In this way, the user could get the realistic visual feedback during progressive surface cutting, and subsequently could focus on the exploration of the internal structures.

Secondly, this thesis proposes a method to simulate the bleeding effects during the virtual surgical cutting and bone drilling. Bleeding is a common phenomenon during a real surgery, and blood provides a very important and unique visual cue for the surgeon. Hence the inclusion of bleeding in a surgical simulation system is essential for a novice doctor to be well trained. The bleeding effect is implemented using a light texture based approach. It could create a realistic visual effect in real time without introducing major additional computational cost to the CPU by utilizing the parallel many-core architecture of the GPU for both the computation and rendering of the fluid.

This thesis implements and evaluates a system that integrated the two proposed modules to perform the task of simulating surgical cutting and bleeding. A haptic device is incorporated to allow the user to interact with virtual objects, and the force feedback is integrated into the system as well. Different force models are designed and implemented in different stage of the whole interactive process of the simulated surgical procedures. The user could not only get the visual feedback, but also feel the haptic feedback, and to be exposed to the full experience of a real surgery as much as possible. Experiments are conducted using openly available resources such as H3D and VHTK on some real patient CT images and several surgery scenarios are simulated. Our results demonstrate the great potential for designing plausible surgical simulator with complex fluid effects in real-time, particularly for virtual reality simulator with kinaesthetically interaction.

Acknowledgement

I would like to thank many people who have made this thesis possible and who have made the last four years a stimulating and very enjoyable experience.

In particular, I would like to express my sincere gratitude to my supervisor, Professor Ling Li at Curtin University, for her generous guidance and invaluable advice. I would also like to express my thanks to my co-supervisor Andrew Squelch for his excellent comments and valuable discussion.

I must thank all the people on the H3D Forum who made their software (H3DAPI and VHTK) available for free on the Web, and always provide invaluable solutions to my problems and confusions with great patience. In addition, special thanks must go to the China Scholarship Council (CSC) and Curtin International Postgraduate Research Scholarships (CIPRS) for providing me with a scholarship, without which I could not have accomplished this project.

I greatly thank my colleagues, En Peng, Li Zhang, Yi Zhang and Gongqi Lin, for their great support and friendship during the four years in the Computing Department of Curtin University.

Finally, I would like to extend my gratitude to my family for allowing me to talk endlessly about the difficulties as a PhD student. Thank you to my parents who are always my strongest backing. Last but not least, I would like to say ‘thank you’ to my husband, Derek, who has always been encouraging and supportive with love and great passion throughout the period of my studies, and has helped me a lot with the formatting of this thesis.

Contents

Chapter 1 Introduction

Apprenticeship-based model has been the major model for surgical trainings for centuries. Experience and skills are obtained over time under the supervision and in collaboration with attending physicians, seniors, and peers. The model is now fundamentally threatened, however, by the numerous concurrent but divergent trends in the social, economic, and regulatory environments of medical training and practice. Economic demands have required increased throughput and efficiencies in the provision of care and have diminished the resources available for surgical training. The transfer of knowledge in an apprenticeship relationship is a time-consuming and resource-intensive process, and as a result that process is under pressure from the need for increasing clinical productivity. In addition, increasing social and regulatory scrutiny has mandated more direct involvement of senior physicians in all aspects of patient care, with the secondary consequence of diminishing the opportunities for responsibility for junior trainees. Although arguably increasing the safety and quality of patient care in the short-term, such changes have potentially dire long-term effects should they result in overall degradation of surgical training.

In this context, surgical simulation is one emerging technology that may help to alleviate this strain. The most rudimentary form of simulation, and the one with the greatest historical precedent, is the use of human cadavers or animal models. However, the supply of available cadavers is limited, and the procurement and maintenance of cadavers for surgical study is expensive. Besides, the use of animal is controversial, and there are ethical barriers to the extensive use of animals in surgical training. Meanwhile, the widespread availability of computers of exponentially increasing capabilities has opened an entirely new field for surgical simulation, in which the simulation is based on a virtual reality, with varying degrees of immersion and realism.

The potential for computers to serve as simulation platforms was initially explored by the military, which first implemented computer simulation in training exercises for astronauts and pilots; both of these roles require extensive training to respond to technically challenging environments with minimal room for error. These early experiments led to the development of the sophisticated, immersive commercial flight simulators in use today, which have sufficient realism to replace or supplement actual flight experience in the training and certification of commercial airline pilots. The success of simulation in this venue both inspire and compel us to implement similar technologic reforms in surgical training, as the standards for competency in patient care should be no less than the standards for competency required of pilots. The rapid advances in hardware, particularly in computer processing power and technological interfaces, make this an ever more attainable goal.

With the development of Computer Graphics (CG) and Virtual Reality (VR), computerized surgical simulators have emerged as a solution to alleviate some of the difficulties previously mentioned. VR systems use computer generated instruments through specially designed interfaces to manipulate computer generated objects. A VR-based surgical simulator is actually a human-computer interface consisting of a network of high-end hardware and software components. Providing a realistic training environment in which trainees act as if they are operating on an actual patient is the simulator's essential goal. An attractive feature of VR surgical simulators is that they can provide objective and repeated measurements, such as the time taken to complete a task, the errors made in the process and also the efficiency with which the movements were made in the accomplishment of the task (Haque and Srinivasan, 2006). These metrics present the opportunity for the assessment of competency without the need for an observer to be present. Suitable tool-tissue interaction models (Misra *et al.*, 2008) coupled to haptic devices allow trainees to feel how much force needs to be applied for different manipulations and to learn hand-eye coordination for specialized techniques (e.g., microsurgery, robot-assisted surgery, or endoscopy). Rare variations of the anatomy can be generated at will and dynamic elements such as breathing, heart beating, and ambient sounds can easily be integrated for enhanced realism. By using surgical simulators, a procedure can be created for preoperative evaluations and beforehand practices for surgeons. The procedure can be recreated or repeated in a virtual environment without harming patients or placing the patient at risk of trauma or injury.

Over the past decades, training systems based on VR technology for different types of surgeries have been developed, including facial plastic surgery (Lee *et al.*, 1999), cataract surgery (Choi *et al.*, 2009), neurosurgery (Wang *et al.*, 2006), knee surgery (Chen *et al.*, 2001), laparoscope (Li *et al.*, 2008), hysteroscopy (Zátonyi *et al.*, 2005), and bone dissection (Morris *et al.*, 2004). Among them, one significant example is the laparoscopy simulator. This is a virtual reality application in surgery simulation for minimally invasive surgery. The technique provides training systems for surgeries to avoid serious damage to the surrounding tissue of the actual area of medical significance. The operation introduces a fibre optic camera and surgical tools through a small number of portals in the skin without cutting through muscles and other tissues lying in the way of the organs or joints of interest. The surgical procedure is performed by operating the instruments through the portals using visual feedback from the inserted camera. Minimally invasive surgery has been proved to be beneficiary in reducing unnecessary tissue damage, physical pain, and recovery time.

Despite the recent successful development of the surgical simulator, there are still new fields of research and there still remains gaps in the field that need to be covered to provide a more realistic surgical simulation, including every natural phenomenon involved in the real surgery. It is a very challenging task especially given the fact that it draws on multiple disciplines and research areas. One of the ongoing challenges in the production of virtual reality environments is the virtual tissue modelling. A variety of computational physics methods have been developed for realistically modelling the virtual surgery environment (Chanthasopeephan *et al.*, 2007; Sokhanvar *et al.*, 2008; Santhanam *et al.*, 2008; Tahmasebi *et al.*, 2008) and the user interactions, including the collision detection and response between soft tissues and instruments (Li *et al.*, 2008), deformation of organs in contact with surgical instruments (Zhu and Zhou, 2010), surgical operations including needle insertion (Chentanez *et al.*, 2009), cutting and sewing (Zhang *et al.*, 2009), etc. A fast and robust deformable model continuously providing visual and haptic feedback to users is essential to the surgery training systems. There are four basic requirements for a deformable model used in real-time surgical simulation: accuracy, efficiency, robustness, and easiness to integrate into the system. In contrast to deformable models used in video games and animations, the purpose of such models in medical simulation is to model the behaviours of realistic biological tissues. The deformation should be controlled by real material parameters such as Poisson Ratio and Young's Modulus taken from biomechanics experiments instead of intuitively adjusted parameters. Some specific deformation effects of biological soft tissues, such as viscous response and free vibration, need to be simulated too. Besides, the models need to be fast enough to provide results within 1/30 to 1/50 second. This eliminates most of the standard methods developed in computational physics, which are too slow to be used in real-time applications. Furthermore, the models need to be robust to provide results under large deformations and in large time steps. Last but not least, the models should be easy to integrate into a complex surgical simulator. In a typical virtual surgical environment, objects with different material properties, phases, geometry shapes and data representations are often combined together, and different types of interactions and responses between them need to be handled together.

Cutting simulation is the key component in surgical simulation as cutting is one of the most common operations in both the conventional and minimal invasive surgeries. Most surgical tasks begin with an incision to expose the surgical region. It is a challenging task to simulate cutting operations since the object topology is changing in real-time and a large amount of computation is usually required for realistic cutting simulation.

The addition of a blood and fluid component to surgical simulation can enhance the user’s experience and aid in learning the procedures and the anatomy. In the surgery, rupture of a major blood vessel causes a great deal of bleeding, and steps must be taken immediately to correct the error. Different kinds of fluid are commonly sucked away or drained from the operation area from time to time. This does not happen through the current teaching techniques using cadaveric bones, but can be replicated using computer simulation. Due to the fact that real time fluid dynamic visualization is computationally expensive, most existing VR surgery simulators tend to avoid involving the fluid dynamic model. However, with the availability of high performance graphic hardware, the improvement of visual quality and computational accelerations become easier to achieve.

To achieve realistic and immersive virtual environments, it is necessary not only to create visually realistic virtual environments but also to create virtual haptic representations that respond accurately to manipulation and surgical manoeuvres. One additional challenge is that, although visually realistic environments can be created at a temporal resolution of 30 Hz, convincing haptic feedback requires a much higher resolution of at least 1000 Hz.

Given virtual surgical simulation is such a complex task, this thesis focuses on progressive cutting and bleeding simulation for surgical operations. In this work, a hybrid cutting model with haptic force feedback has been developed to be joined into a surgical operation simulator. To greatly enhance the realism of the surgical simulation, a real-time fluid model is integrated with the surgery cutting for simulating the bleeding effects. To achieve high performance results, the fluid model utilizes the GPUs capable streaming computations. The blood will flow on the anatomical surface from the cutting path when the surgical knife cut the surface to a certain thickness. The prototype employs a portable system to run a simulator on a standard personal computer with graphic processor units (GPUs). A high fidelity of force feedback device with specialized software also enhances the establishment of a more realistic simulator with real-world characteristics.

1.1 Aims and Approaches

1.1.1 Aims

The main goal of this thesis is to develop a VR based surgical simulation system that involves cutting and bleeding effect. The virtual tissue modelling should take biomechanics properties of human anatomy and organ into consideration. The system may provide a high-fidelity graphic display and realistic haptic feedback in real time during a synthetic surgical operation. The user could play with the system to practice the operation, with the added complexity of real time fluid flow phenomenon of the blood during progressive cutting. The interior structures and material properties would also be revealed during the cutting process to provide visual clue for the users. The expected research outcomes can be broken down as follows:

1. **Progressive cutting based on a hybrid model for the anatomy objects**. Propose an approach of realistic smooth surgical cutting on deformable anatomy objects. The modification of topology should not lead to large amount of new elements addition. The internal structures and material properties of heterogeneous objects should be taken into considerations.
2. **Fluid Model**. Develop a method for real time fluid simulation to simulate bleeding effect during surgical cutting for realistic visual effect. No major additional computational cost should be introduced to the CPU as enormous

computational load has already been placed on the CPU for simulating deformations, cutting, collision detection and response characteristics.

3. **A prototype of surgical simulator**. Establish a PC based surgical simulator that resemble the real-world surgical operation for cutting and drilling etc. Apart from the creation of visually realistic virtual environments, haptic device and force feedback should also be integrated into the simulator for high fidelity. The fluid model is to be integrated into the simulator to build a realistic surgical simulation that resembles the real surgery.

1.1.2 Approaches

The requirements for a real time surgery simulator are quite demanding. The system is expected to manage simultaneously the object deformation (cutting and drilling), fluid simulation, visual rendering for display, and haptic rendering for force feedback. Beside the computational intensiveness in object deformation and fluid computation, the system has to handle the required threads execution between a slow and high frame rate: slow frame rate (30Hz) for the display and high frame rate (1000Hz) for the haptic force computation. Both of them must be synchronized to provide high fidelity surgical simulation. Such demanding tasks are usually achieved by very powerful computers such as a supercomputer. However, involving supercomputer makes the system expensive and not portable, generally not applicable to an ordinary user.

By addressing limitations in previous surgical simulator systems, this thesis proposes several approaches to implement surgical operations such as cutting and drilling with fluid simulation on a standard PC. The distinct approaches can be identified as follows:

1. A hybrid model consisting of an inner volumetric model and an outer surface model is proposed to simulate surgical cutting on deformable anatomy objects. A node-snapping technique is presented to modify the surface topology of the objects, without adding new elements. Progressive smooth cut is generated by duplicating and displacing mass points that have been snapped along the cutting path. A volumetric deformable model is employed underneath the surface with considerations on the internal structures and material properties of heterogeneous objects. A cutting gutter is generated by 3D ChainMail

deformation applied on the volumetric mode. A seamless connection is built between these two data models so that the volumetric gutter is formed according to the outer surface cut opening progressively.

2. The fluid model is a texture based approach that uses pixel and vertex shader level techniques in the GPU. By utilizing the parallel many-core architecture of the GPU for both computations and rendering of the fluid, the CPU could be freed up for other time-critical tasks like deformations, collision detection and collision response and the whole system could achieve real time performance.
3. The prototype of surgical simulator is implemented using the open source H3DAPI (http://www.h3dapi.org) and Volume Haptics Toolkit (VHTK) (http://www.h3dapi.org/modules/mediawiki/index.php/Volume_Haptics_Toolkit). H3DAPI is an open source haptics software development platform that uses the open standards OpenGL and X3D with haptics in one unified scene graph to handle both graphics and haptics. VHTK extends H3DAPI by introducing the scene-graph nodes necessary for loading volumetric data, handling and processing the data and for using the data to produce both visual and haptic feedback. Our prototype could provide high-fidelity graphic display and realistic haptic feedback during the surgical cutting and organ deformations interactively.

By blending the three approaches above with collision detection, force calculation and efficient visualization, a surgical simulator could be achieved with smooth progressive cutting and realistic fluid dynamic simulation in real time.

1.2 Structure of the Thesis

This thesis is organized as follows. In Chapter 2, a review of related work in the fields of virtual surgery simulation is presented. The overview of existing virtual surgery simulators is first briefly presented. As a computer-based surgical simulation system draws on multiple discipline and is such a complex research area, only some selected technical themes closely related to the research goals are presented. Next the sources and pre-processing of the data used in this thesis are discussed. This is followed by discussions on virtual tissue modelling which could generally be categorized into physically-based and geometry-based approaches. Surface cutting approaches are then reviewed based on how their solutions address the following issues: definition of the cut path, primitive removal and re-meshing, number of new primitives created, when re-meshing is performed, and representation of the cutting tool. Previous works in volume deformation are then discussed, focusing on the linked volumetric representation. Finally, a detailed review is conducted on bleeding simulation approaches, which are divided into physically-based approaches and non-physically-based.

In Chapter 3, the surface deformation is first examined. A progressive surface mesh cutting technique is then discussed in details, which is based on the work by Lin *et al*. (2007). The deformation of the incision is also discussed for wrinkle effect during the pulling process. A basic structure of volumetric data and the visualization of volume data are later reviewed. A groove generation based on ChainMail algorithm is implemented with the novelty of unlinking the neighbour elements and deforming each side to simulate the shape of groove and to present the interior structures of the heterogeneous objects. A drilling effect is also achieved to show the extendibility of our volumetric model. Finally, our hybrid cutting model is built which combines the surface model and the volumetric model. By building a tight connection between them, the progressive cutting could be achieved simultaneously on these two data models seamlessly.

A general introduction of haptics interaction is also given with some detailed explanations of force calculation in Chapter 4. Firstly the roles of haptics on surface manipulation are discussed and the force models are explored in details. The haptics on volumetric model is then exploited for more realistic force feedback.

A lightweight fluid model is proposed in Chapter 5. Our low-cost method for generating realistic bleeding effect in VR-based surgical simulators outsources the computations to the GPU, thus freeing up the CPU for other time-critical tasks. This method is independent of the complexity of the organ models in the virtual environment. An effective hierarchical tree data structure is also generated to manage the geometry to enhance the speed of object manipulation and keeps the computation load stable.

Experimental results and discussions are provided at the end of Chapter 3, 4 and 5; specifically demonstrate the effectiveness of the proposed methods.

Finally, Chapter 6 provides a summary of the thesis, its contributions and potential future directions in virtual surgery simulation.

Chapter 2 Literature Review

Surgical training has traditionally been one of apprenticeship, where the novice surgical trainee learns to perform surgery under the supervision of a trained surgeon (Gurusamy *et al.*, 2009). The operating room and the patient, however, comprise the most common, the most readily available, and often the only setting where hands-on training takes place. The novice surgeon acquires skills by first observing experienced surgeons in action and then by progressively performing, under varying degrees of supervision, more of the surgical procedures, as his/her training advances and his/her skill level increases. Different procedures have different learning curves (Herrell and Smith, 2005). Surgeons experienced in one procedure may not be experienced in another, and results improve with experience in an individual procedure.

The drawbacks of this traditional training approach have been underscored in the 2000 report by the Institute of Medicine, entitled "To Err Is Human: Building a Safer Health System" (Kohn *et al.*, 2000) which points out that about 100,000 deaths per year in the United States occur as a result of medical errors, making it the eighth leading cause of death. A three-year study by HealthGrades (Golden, CO, USA, 2008), an American healthcare ratings organization, also found that medical errors resulted in over 230,000 deaths in American hospitals during the study period. In a different study (Vickers *et al.*, 2009) based on rates of cancer recurrence in 4,700 patients operated upon using keyhole techniques by 29 surgeons in seven hospitals throughout Europe and North America, Vickers *et al.* reported that surgeons require 750 operations to perfect keyhole surgery procedures.

As technology progressed, many different tools and techniques have been deployed to provide added value to the medical training process such as anesthetized animals or cadavers (Coles *et al.*, 2011). However, animal models and cadavers are not considered good substitutes for human patients due to the fundamental differences in anatomy and tissue consistency. Not only are cadavers expensive, but procedures can only be performed once and a mistake can render the body useless to re-demonstrate a procedure. Moreover, the use of animals and cadavers for training purposes also raises ethical issues.

Therefore, relatively inexpensive inanimate training methodologies such as video tool box trainer are becoming more popular. The Society of American Gastrointestinal Endoscopic Surgery (SAGES) is the first organization that has officially adopted an inanimate training methodology and established the Fundamentals of Laparoscopic Surgery (FLS) course (Peters *et al.*, 2004), which is a comprehensive program designed to teach the cognitive and psychomotor aspects unique to laparoscopic surgery. The overall goal of the FLS program was to "teach a standard set of cognitive and psychomotor skills to practitioners of laparoscopic surgery" in the belief that knowledge and application of these fundamentals would help "ensure a minimal standard of care for all patients undergoing laparoscopic surgery." The FLS program consists of two components: a didactic module for education an examination to assess competency. The didactic learning modules are CD-ROM based and teach the underlying physiology, fundamental concepts, and component manual skills involved in laparoscopic surgery. The assessment instruments were designed to be in accord with the competency movement and the American Board of Medical Specialties recommendation for maintenance of practice and practice-based learning and improvement.

However, both the animate and inanimate training techniques suffer from the same drawbacks, the most important being the need for an instructor/supervisor, non-standardized methods of feedback, inadequate provision to practice for rare medical conditions, and lack of well-defined subjective methods of performance evaluation. These training methods and trainers have access to a limited number of parameters and often time is the only performance measure.

These drawbacks have prompted the development of VR based surgery simulators where the human user is able to interact with three-dimensional virtual models of organs using his/her sense of vision as well as actively manipulate them using his/her sense of touch through a haptic interface device such as a PHANToM from Sensable Technology^{TM}, It is well recognized that VR based simulation systems offer a unique way of objectively assessing performance while imparting training, providing real-time feedback, tracking the trainee's learning curve over extended periods of time, and providing quantitative scores while, at the same time, offering summative evaluation during examinations .

A computer-based surgical simulation system draws on multiple disciplines. It has both technical and cognitive aspects. The technical components include a virtual-patient model and specialized input and output devices. The model must demonstrate physical properties consistent with a live patient. Thus, virtual soft tissues should deform with contact pressure and should have the same texture and consistency as live tissues. The visual representation of the region of interest must be consistent with intra-operative views, and tissue perfusion and appearance must be consistent with actual scenes. Creating a virtual model of the relevant human anatomy draws on research from disparate fields, such as computer science and bioengineering.

A simulator engages the surgeon through multiple sensory channels. The surgeon's actions are tracked and replicated in the virtual operating environment. Visual and haptic effects such as bleeding and tissue resistance are rendered on the appropriate hardware. These devices draw on research on visual and haptic displays.

An effective simulator integrates its technical components with medical and educational content designed to impart specific skills or knowledge to the user. A simulator's cognitive components draw on research on learning theory, performance measurement, and surgical knowledge.

As surgical simulation system is such a complex research area, it is unrealistic to cover all the relevant fields in this thesis. Therefore, only some selected themes are presented in this literature review. They are: Data sources and pre-processing, Soft tissue modelling, Surface cutting, Volume deformation, Fluid simulation and Haptic interaction. The following sections will serve as an overview of the technical themes involved in the remaining thesis and will outline the focus of the thesis.

2.1 Overview of Virtual Surgery

Virtual surgery is a type of simulation that can be used to practice the often dangerous surgical procedures without the need of an actual patient. This technique refers to a virtual reality simulation of surgical procedures and is used as an analogue for the actual surgery where doctors can practice on a virtual patient before performing the surgery. Virtual surgery is usually associated with visualizations of an operated organ on a three-dimensional (3D) graphic display. The user can interact with the simulated 3D objects, such as organs and tissues, through a force feedback device that resembles various surgical tools to physically feel, touch, manipulate and operate on the objects. The 3D patient data generated from computerized tomography (CT) and magnetic resonance imaging (MRI) scans can be reconstructed as 3D anatomy objects in the virtual environment. Surgeons can practice the operations multiple times without the use of cadavers or animals. Virtual surgery can simulate real world surgical scenarios to allow for intensive training activities. This technology would let virtually trained doctors and medical students to be more proficient and get a better preparation with fewer errors.

The simulation of surgical procedures using virtual anatomical models is a rapidly growing field in medical imaging. This is due to the availability of virtual reality techniques, and also because of the availability of detailed virtual anatomical models. Surgical simulators have been developed for a wide range of procedures. They can be broadly classified in terms of their simulation complexity. Three classes are considered: needle-based, minimally invasive, and open surgery (Liu *et al.*, 2003).

2.1.1 Simulators for Needle-based Procedures

As the name implies, needle-based procedures deal with the task of controlling the insertion of some needle-type instrument into the human body. Examples include vascular access, catherization, biopsy and anesthesia. There can be some problems when surgeons manually insert needles, including different treatment results and the possibility of a complicating disease resulting from needle insertion into large blood vessels, etc.

Some of the first prototypes of needle-based procedures focused on the haptics and to a lesser degree visualization and general simulation of tissue. Brett *et al.* (1997) developed a simulator for surgical needles with haptics only, without any additional visualization. The CathSim simulator developed by Tasto *et al.* (1999) has both very realistic graphics and custom haptic. In the CathSim the users can feel a realistic "pop" from the puncture of a vein inside the skin. The visualization supports different ages, skin colour and health condition. This project was commercialized as the CathSim AccuTouch System by Immersion Medical. Cotin *et al.* (2005) presented a recent high-fidelity simulator for interventional radiology, in which a guide-wire catheter is inserted into the artery network of a patient, monitored through live x-ray images. The simulator includes real-time deformable models and handles the many concurrent collision points between guide-wire and vessel. The visualization simulates x-rays through the deformed tissue resulting in realistic x-ray images. Zhu *et al.* (2007) presented a training system designed to improve the skills of interventional radiology trainees in ultrasound-guided needle placement procedures. Hauser *et al.* (2009) presents a feedback controller that steers a needle along 3D helical paths, and varies the helix radius to correspond to perturbations. Their controller was evaluated under a variety of simulated perturbations, including imaging noise, needle deflections, and curvature estimation errors and proved to be able to reduce targeting error by up to 88% compared to open-loop execution. Kobayashi *et al.* (2010) aims to develop an integrated robotic system with image guidance and deformation simulation for the purpose of accurate needle insertion. In vivo experiments were also conducted to verify the effectiveness of their needle insertion manipulator. Goksel *et al.* (2011) presents a haptic simulator for prostate brachytherapy. Both the needle insertion and the manipulation of the transrectal ultrasound probe are controlled via haptic devices.

Needle-based simulators generally have limited degree of freedom in interaction and the demands for visualization are in many cases quite limited. One area of specific interest for these simulations is the haptic feedback, which in many needle-based procedures is the key to a correct execution. Once the needle is inserted into the patient the interaction is often restricted to 1 DOF in the direction of the guidance of the needle. In some cases special-purpose haptic equipment has been built taking into account this limited interaction, but in many cases general 6 DOFs equipment has also been used. They are useful for teaching relatively straightforward procedures with well-defined algorithms. Their simplicity makes them widely performed procedures at low cost. In situations where opportunities for practice are limited or where current methods using animal models are not optimal, needle-based simulators could be very useful.

2.1.2 Simulators for Minimally Invasive Surgery

Minimally invasive procedures use specially designed instruments. The instruments are inserted into the body via small incisions. Visual feedback is obtained via inserted scopes, cameras, or fiberoptic devices, and a video display monitor is used to show the image. Because the entry portal is small, these instruments have a limited range of motion. For example, laparoscopic instruments are constrained to pivot about the entry port on the abdominal wall. Other instruments are designed to work in confined areas as well. For example, only the tips of bronchoscopes can be flexed under user control. In both cases, haptic feedback is muted due to sealing gaskets (such as in laparoscopic instruments) or the length of instrument within the patient's body (as with bronchoscopes).

The limited range of motion and haptic feedback, and the use of specialized tools and video displays facilitate the development of simulator in minimally invasive surgery. The first laparoscopic VR simulator developed for teaching technical skills was the Minimally Invasive Surgery Trainer-Virtual Reality (MIST-VR) (Mentice, Gothenburg, Sweden) in 1997 (Wilson *et al.*, 1997). The MIST-VR is comprised of a laparoscopic interface with motion-tracking devices attached to mock laparoscopic instruments. This is attached to a computer that displays the movements of the instruments in real time on a computer monitor (Aggarwal *et al.*, 2004). The MIST-VR consists of abstract psychomotor tasks, which allows it to be sued for training a wide range of surgical specialities in basic endoscopic techniques, including cardiothoracic, general and gynaecological surgery. However, it does not allow training in specialist techniques that are provided by procedural and full length tasks on other simulators.

The LapSim (Surgical Science, Gothenburg, Sweden) laparoscopic simulator was developed in 2000. It was developed with more realistic tasks using tissue that is be manipulated and could bleed. Tasks such as “clip and cut” and “suture” introduced tasks that are more relevant to a surgeon. The LapSim was one of the first VR simulators that allowed students to practice parts of operations such as laparoscopic cholecystectomy and gynaecological procedures. The LapSim is used for training and assessment of both surgery and gynaecology trainees (Aggarwal *et al.*, 2006; Hart *et al.*, 2007).

The LAP Mentor surgery simulator (Simbionix, Chicago, USA) was developed in 2002, which provides full length operations range from basic to advanced laparoscopic procedures. The LAP Mentor also combines the extensive range of procedures with advanced haptic feedback hardware that allows the simulator to transmit resistance when tissues or objects are encountered during a simulated task.

This enhances the realism of the simulator, and as such is used in another laparoscopic simulator (Aggarwal *et al.*, 2009).

At the 2006 Society of American Gastroesophageal Surgeons (SAGES) meeting, 63 attendees at the SAGES Learning Centre were invited to evaluate a new virtual reality robotic surgery simulator (SEP Robot; SimSurgery AS, Oslo, Norway) as well as either a computer-enhanced laparoscopic simulator (ProMIS; Haptica, Ltd., Dublin, Ireland) or a laparoscopic virtual reality simulator (SurgicalSIM/Surgical Education Platform; SimSurgery; and METI, Sarasota, FL). Subjects were assessed during one iteration of laparoscopic suturing and knot-tying on the SEP Robot and either the ProMIS or the SurgicalSIM. A post-task survey determined users' impressions of task realism, interface quality, and educational value. Performance data were collected and comparisons made between user-defined groups, different simulation platforms, and post-task survey responses. The results have shown that with the help of SEP Rbot and either ProMIS or SurgicalMIS, it is possible to limit patient risk and to increase training efficiency for minimally invasive procedures.

Among all minimally invasive surgery simulators, those for laparoscopy and endoscopy are the most advanced. The interior anatomy generally contains sufficient details and realism for educational purposes. Commercially available laparoscopic trainers can teach basic skills such as camera navigation, grasping, suturing and knot tying, and cauterization. There are extensive evidences to support the validity of laparoscopy simulators for both training and assessment of technical skills (Van Dongen *et al.*, 2007; Bajka *et al.*, 2010) Advantages of laparoscopic surgery include decreased morbidity, reduced costs for society (less hospital time and quicker recovery), and also improved long-term outcomes. Despite recent advances, limitations still exist. Surgical effects, such as bleeding, blood pooling, and tissue tearing, are presently simulated with limited realism. Real-time tissue and organ deformation are generally limited to specific organs or simple structures such as arteries, ducts, and other tubular structures. For these and other reasons, the goal of simulating medically relevant procedures from start to finish has not yet been reached.

2.1.3 Simulators for Open Surgery

Open surgery requires larger incisions on the body. The surgeon often has direct visual and tactile contact with the region of interest. The visual field, range of haptic feedback, and freedom of motion are considerably larger compared to minimally invasive procedures. Open surgery is thus more difficult to simulate.

Webster *et al.* (2001) presented a PC based suturing simulator for wounds. The system supports haptic feedback through Phantom Omni devices. Bielser *et al.* (2002) presented what they called an "Open Surgery Simulator". The system could basically simulate interaction between skin and surgical hooks or surgical scalpels, but without any abdominal content. Gasson *et al.* (2004) presented a system for hernia repair, supporting knives and clamps. The system used a non-linear Spring-Mass system as the basis for haptic interaction. The system used Phantom interaction devices. Nakao (2003) proposed his ActiveHeart system where one can practice the initial incision into the chest and palpation of the aorta in conjunction with a pure visualization of the heart. The later work by Mosegaard (2006) fits nicely into Nakao's research as they investigate the actual surgical procedure after the opening of the chest. Wang *et al.* (2006) focus on the simulation of surgical prodding, pulling and cutting. Advanced features such as the separation of the cut surfaces by retractors and post-cutting deformations are also included. Their experimental results are enhanced by implementing 3D stereo-vision and the use of two hand-held force-feedback devices. The work by Pang *et al.* (2007) focuses on the orthopaedics surgery training system with components for modelling, for simulating the deformation, and for visualization in an efficient way. By accelerating the computation with the Physics Processing Unit (PPU), a high fidelity of realism and interactive frame rate can be achieved in a virtual environment simulating both soft-tissue deformation and bleeding. However, their visualization of blood and soft tissues need to be enhanced with advanced rendering techniques. Beside, the use of PPU restricts the scalability of their method. Later, their work is extended into a collaborative simulation system based on the cluster-based hybrid network architecture by Qin *et al.* (2009). Multicast transmission is employed to transmit updated information among participants in order to reduce network latencies, while system consistency is maintained by an administrative server.

Though minimally invasive procedures have become the standard of care for many types of elective thoracic and abdominal surgery, it is imperative for surgical residents to have both the confidence and the technical skills to safely and effectively perform open surgical procedures when the need arises. When adverse event occur, either as a primary problem or as a complication of a minimally invasive operation, and cannot be safely or expeditiously dealt with using minimally invasive techniques, the standard of care dictates that an open procedure be undertaken. The design of open surgical simulators still poses a great number of technical challenges: appropriate soft tissue models have to be chosen and the underling differential equations have to be solved efficiently. Very often, surgical hooks and large scalpel intersections need to be applied simultaneously, featuring complex interactions with the soft tissue. High speed haptic rendering of these tools requires sophisticated mechanical models. Considerable advances in haptics, real-time deformation, organ and tissue modelling, and visual rendering must be made before open surgery can be simulated realistically, so that these simulators could focus on gaining experience in the identification of open anatomy, achieving adequate intraoperative exposure, performing surgical dissections efficiently and safely, and controlling major intra-abdominal bleeding.

2.2 Data Sources and Preprocessing

The type of objects dealt with in this thesis is related to the shape and physical characteristics of organs and tissue. To build realistic virtual models we need to base them on real human anatomy data. Currently the most important techniques to acquire medical images from human bodies are:

- **Computer Tomography (CT):** The CT x-ray beams are used for imaging. The body is moved through a tube while a rotating x-ray is taking images of cross sectional slices. For each angle the amount of radiation not absorbed is measured. The measurements from different angles are used to compute the content of the slice with a method called filtered back projection. The resulting value for each discretely sampled point of the volume is directly proportional to the density of the scanned tissue. The density is measured in the so-called Houndsfield Units. The Houndsfield Scale starts with – 1000 HU (radio density of air). Distilled water has a value of 0 HU. The scale has no maximum value but in reality it is limited to a range 12bit (-1024 to +3071).

The downside of this technique is the exposure of patients with radiation.

- **Computer Tomography Angiography (CTA):** A disadvantage of the CT method is the low contrast between blood vessels and other soft tissues. To overcome this problem a contrast agent can be applied.
- **Magnetic Resonance Imaging (MRI):** MRI exploits the fact that protons have a so-called spin and nuclei have magnetic properties. With a combination of strong magnetic fields and radio waves the orientation of the spin axis can be manipulated. If one of the magnetic fields is switched off the protons react with a relaxation rotation back to the initial spin axis orientation. This movement induces a measurable magnetic field which is used to compute the volume data.

  The measured value is the relaxation time. Two different relaxation movements are measured.

  1. **T1-weighted imaging** measures the longitude relaxation time. T1-weighting needs longer relaxation times (>1000ms) but generates a higher spatial resolution.
  2. **T2-weighted imaging** measures the traverse relaxation time. The relaxation times are relatively short (<100ms). T2-weighting leads to better contrast between different tissue types.

  MRI imaging provides no scale for the measured value like the Houndsfield scale. Different values have to be interpreted relatively. In contrast to images generated with the CT method, MRI allows a much better discrimination between different tissue types.

  The drawback of this method is that results are more noisy than images generated through the CT method.

- **Magnetic Resonance Angiography (MRA):** As in CTA, magnetic resonance angiography emphasizes the blood vessels. Blood vessels can be emphasized either by some adjustments and enhancement on the MRI method or alternatively a contrast agent can be used.

Often the datasets are divided into two categories according to the types of usage: patient-specific or general datasets. Working with acquisition of real patient datasets means working with real patients and therefore a number of constraints apply. Firstly, the time available to acquire the datasets and the variety of modalities possible are limited by the comfort and safety felt by the patients. Secondly, the total resources used for a large number of patient-specific datasets limit the amount of manual work that can be put into each dataset. Very long post-processing times are often unacceptable either, since this could delay the treatment process of each individual patient. General datasets can be acquired from volunteers, in which case time and resources are not a serious problem. The most aggressive acquisition of general datasets can be performed when deceased donate their bodies to medical research. In such cases long post-processing times, high-dose radiation, and destructive acquisition can be allowed. The prime example of this is the Visible Human Project (Ackerman, 1998) where a donated male body has been acquired with CT, MR and cryosection (slicing of the body in millimetre thick slices and photographing each slice). A number of projects following the Visible Human Project have arisen; the Chinese Visible Human (Zhang *et al.*, 2003), the Korean Visible Human (Park *et al.*, 2005) and the Visible Ear (Sørensen *et al.*, 2002), just to name a few.

Preprocessing may include segmentation and surface extraction. Segmentation is the process of dividing the image data into regions according to some measure. In the case of medical image segmentation it seeks to identify the different parts of anatomy. A surface can then be extracted from the segmented medical image data through e.g. the Marching Cubes Algorithm (Lorensen and Cline, 1987). The output surface mesh may then be processed by a modelling program such as ParaView to smoothen or resample. The medical images can also be directly used for direct volume deformation (DVD) and direct volume rendering (DVR).

2.3 Soft Tissue Modelling

Simulation of procedural tasks has the potential to bridge the gap between basic skills training outside the operating room (OR) and performance of complex surgical tasks in the OR, but it requires realistic modelling of soft tissues in an intra-operative situation. Real-time monitoring of the forces exerted on an organ due to manipulations by a surgical instrument during an operation would allow the determination of local tissue stress. In this way, when damage thresholds for tissue stress are known, safety margins can be imposed upon the instrument to avoid irreparable tissue damage. An important issue for realistic modelling of the intra-operative situation is the proper representation of the manipulated tissues. This implies the characterisation of the in vivo mechanical properties of the tissue and subsequently implementing these properties in a mathematical model of the organ in question.

In computer graphics, different approaches are discussed for soft tissue modelling. For deformable soft tissue modelling, two approaches are present in the literature: physically-based and geometry-based models. In geometry-based techniques, the object or surrounding space is deformed by manipulating vertices or control points. These techniques are relatively faster and easier to implement. However, they do not capture the physics of the problem. The physically-based techniques, on the other hand, aim to model the physics involved in the motion and the dynamics of interactions.

2.3.1 Geometry-based Techniques

Many methods exist as geometry-based techniques for soft tissue modelling, but deformable spines have been the most popular in this branch. In general, splines serve to obtain smooth and rounded curves, surface, or volumes, which can be adjusted through a series of control points. By moving the control points, the form of the respective curve, surface, or volume changes accordingly. Among the different existent techniques are the well-known Bezier curve and NURBS (non-uniform rational B-Splines). Deformable splines, also known as active contours, were the first deformable models (in the strict sense) to be developed (Terzopoulos *et al.,* 1988), and they were also the first models to be applied to the field of surgery simulation (Cover *et al.*, 1993). As a starting point, they employ classical splines to model a 3D object (or its surface) (Richa *et al.*, 2010). With the fundamental theorems of differential geometry regarding the equivalence of shapes, deformable splines then define a potential energy, which is proportional to the degree of elastic deformation. By using the Lagrange approach, this energy is finally minimized with respect to the displacements enforced in some control points, thus obtaining the corresponding deformation state.

The elevated number of parameters equips this deformable model with a substantial level of control over the shape and the physical properties of the mesh. However, these parameters can only be chosen arbitrarily and are difficult to determine empirically. In addition, the representation of an object as a smooth surface does not coincide with the rendering algorithms of modern graphics cards, which are oriented towards plane polygons. Thus, a further processing step is required. Even without this additional expense, deformable splines already have a very high computational cost. This is why solid objects usually have to be modelled as hollow shell, with the corresponding detriment to realism.

On the whole, deformable splines are more complex and computationally costlier than spring-mass type models, without actually offering better realism. This is why they are hardly employed currently.

2.3.2 Physics-based Techniques

2.3.2.1 Mass-spring Model

The Mass-Spring model employs ordinary differential equation (ODE) systems which use mathematic methods, explicit or implicit, to solve numerical approximations. In this method, a discretized object is used. To simulate the deformation behaviour, motion equations are derived on grid points which are called masses, which store position, velocity and acceleration values. The connections between neighbouring nodes are called springs, which are assigned stiffness and rest size values (Selle *et al.*, 2008; Mesit *et al.*, 2010). Springs tend to keep the system in balance - maintain a safe distance between the points of the masses. The forces acting on each point are dependent on the distance between points adjacent to that point and the external forces acting on the system (e.g. gravity).

The mass-spring model is widely used in surgical simulation. It is relatively easy to implement and does not require high computational power. Visualization and modification of the model in real time is achievable. Unfortunately, this model also introduces important limitations. Deformations are not accurately reproduced; it is only an approximation of the real behaviour of tissues. In addition, delays are introduced due to the propagation of forces in the system. Any modifications to the model, such as removing of grid points or cutting of springs, force adjustment of the system parameters. It also has a tendency to oscillate, due to the iterative method for calculating the forces acting on grid points.

2.3.2.2 FEM

Different from the Mass-Spring model, the Finite Element Method (FEM) works on the continuum mechanic to approximate interpolation functions. In order to use an irregular grid for deformable object simulation, the object is defined as a set of finite element consisting of a set of nodes.

The FEM advantage is that complex geometry, general boundary conditions and variable or non-linear material properties can be handled relatively easily (Kaufmann *et al.*, 2009a; Kaufmann *et al.*, 2009b; Becker *et al.*, 2009). However, a major drawback of this method, especially for nonlinear description, is the significant computation time that may be required to solve for large models.

2.3.2.3 PAFF

The Point-Associated Finite Field (PAFF) approach acts under the partial differential equation system based on continuum mechanic. The object is discretized using a scattered distribution of nodal points or particles that serve as the computational primitives, much like those used in the mass-spring models. Unlike the mass-spring models, the governing partial differential equations are solved. It is important to note that the method is "meshless" since no direct link exists between the computational nodes. Each node has a "region of influence" which smears out their effects and coordinates their motions during simulation.

The PAFF is a newer method compared with the others discussed previously, and it is more efficient over the FEM for forces distribution to the object. The PAFF is able to generate good simulation results in real time, including haptic feedback (Lim *et al.*, 2005; De *et al.*, 2006; Lim *et al.*, 2006).

2.4 Surface Cutting

In the past, cadavers were considered the golden standard of surgical simulation. However, cadavers are of limited supply and are costly to acquire and maintain. The development of alternative training methods is considered important to the future of surgical training (Malone *et al.*, 2010). Surgical simulators provide a no-risk environment where skills can be gained through harmless practice repeatedly. Cutting simulation is the key component in surgical simulation as cutting is a common operation in both conventional and minimal invasive surgeries. Most surgical tasks begin with an incision to expose the surgical region. However, it is a challenging task to simulate cutting operations since the object topology is changing in real-time and a large amount of computation is usually required for realistic cutting simulation.

In surgical simulation, it is generally not necessary to provide a physically-based simulation of the internal forces involved in the deformation, as a huge computational cost would be required. Instead, it is more important to show realistic internal structures at the cutting site as the visual clue, especially for surgical training and planning purposes. Therefore, priority should be given to the realistic surface deformation and smooth cutting before there is an opening. Afterward, the users often focus on the exploration of the interior features.

Even though a great deal of information about the interior structures and material properties of the heterogeneous tissues are discarded, surface-based cutting methods can provide smooth and realistic cutting effect at a low computation cost. Complemented with volumetric models which represent appropriate internal structures and material properties, surface-based cutting methods can have a role for situations in which high-fidelity virtual environments are required.

According to Bruyns (2002), cutting methods can be distinguished by how their solutions address the following major issues:

1. Definition of the cut path,
2. Primitive removal and re-meshing,
3. Number of new primitives created,
4. When re-meshing is performed, and
5. Representation of the cutting tool.

2.4.1 Definition of the Cut Path

Cutting a virtual object requires disconnecting one mesh primitive from another. For an interactive cutting tool, the user must begin by inspecting the object and determining where the cut should begin. After selecting this starting point, the user then defines the shape of the cut path. Existing cutting tools determine the cut path by either:

(a) Placing seed points on the mesh,
(b) Placing a template through the mesh, or
(c) Moving a virtual tool through the mesh.

In the first approach, the user selects successive points on the object's surface. These points determine the basic outline of the cut path, but must be linked together to form a continuous cut through the object. Choices for linking the points include Dijkstra's shortest-path algorithm or constructing successive planes between the points. Using Dijkstra's shortest-path algorithm to link the seed points has the drawback that only existing vertices can be used to define the cut path. Unless a regular grid is used, the resulting contour is frequently jagged. Li *et al.* (2005) proposed a new topological data structure for representing a set of polygonal curves embedded in a meshed surface. In this implementation, the vertices of the curve do not necessarily correspond to the vertices of the surface, thus could obtain smooth cut borders. Moreover, in order to enable more complex cut than a simple curve, they adopted incremental insertion of a new cutting curves, i.e. with every cutting curve designed, new intersections between this new curve and the existing surface should be dynamically found and updated. However, the embedded polygonal curves are predefined, which is not suitable for progressive cutting.

In the second approach (Schutyser *et al.*, 2000), the user has a predefined shape for the cut path represented by another object in the virtual world. The user interactively positions the shape, creating the desired intersection with the object to be cut. When the shape is positioned a signal is made to cut the object.

In the implementation of the third approach, the user moves a virtual tool in the world and the cut path is determined by successive intersections of the tool with the object primitives (Kim *et al.*, 2010; Turkiyyah *et al.*, 2011). But this implementation generally suffers from the increment of the number of elements, which may slow down the computation.

2.4.2 Primitive Removal and Re-meshing

Cutting techniques may also be classified based on the way the cutting operation is implemented. Three common methods are:

(a) Removing intersected primitives,
(b) Mesh subdivision,
(c) Mesh adaptation.

The element removal technique basically removes the elements that are intersected by the cutting tool (Cotin *et al.*, 2000; Choi *et al.*, 2009). Forest *et al.* (2005) proposed an efficient method for removing tetrahedral from a tetrahedral mesh while keeping its manifold property. This algorithm is used in the context of real-time surgery simulation where the action of an ultrasonic lancet can be simulated by the removal of a small set of tetrahedral from a tetrahedralization. Despite of its simplicity and computational efficiency, this method violates the physical principle of mass conservation due to the removal of elements and, more importantly, it is difficult to present a smooth cutting because of visual artefacts and the cut surfaces look unnaturally jagged.

More appealing visual representations of incisions are made possible with mesh subdivision methods (Bielser *et al.*, 2004; Huy Viet *et al.*, 2006). They usually classify a cut according to the different rotational invariant intersection states. Predefine subdivisions of mesh elements are then performed. The deficiency of this method is the increment of the number of elements and the creation of smaller or degenerated elements. The newly increased elements may slow down the computation and the degenerated elements as well as the edges with widely varying lengths may cause the instability of numerical calculations of deformation.

These problems could be ameliorated with mesh adaptation approaches. The main idea is to approximate a cutting path with existing edges or surfaces. It enables mesh incisions without large increase of element count and occurrence of small elements. Mass points near cutting path are snapped to the closest points on the cutting path such that some related edges are aligned along the cutting path. The mesh adaptation method starts with collision detection. After the initial collision of the haptic tool with the object surface, intersection points between the cutting path and the underlying polygon edges are calculated. It is worth mentioning that the marked intersection points are only potential as the cutting is a progressive process. Whenever an intersection point is calculated, the local area of the mesh is updated before moving to the next intersection point. Serby *et al.* (2001) and Nienhuys *et al.* (2001) both proposed mesh adaptation based cutting approaches for very smooth and realistic surgical cutting simulation. Lim *et al.* (2007) presented a hybrid approach to the simulation of surgical cutting procedures by combining mesh adaptation technique with a physically based meshfree computational scheme, the point-associated finite field (PAFF) approach, and empirical data obtained from controlled cutting experiments. To enhance the realism of the rendered scenarios, an innovative way of using images obtained from videos acquired during actual surgical processes was also proposed.

2.4.3 Number of New Primitives or Primitives Created

Cutting techniques can be further classified according to the number of new primitives created during re-meshing. Existing cutting techniques handle re-meshing by either:

a) Disallowing new primitives,
b) Allowing unnecessary new primitives, or
c) Creating a minimal number of new primitives.

In the first method, selecting a subset of the mesh traversed by the tool creates the cut path. The vertices of the subset are then repositioned by moving the edges of the traversed faces to the cut path. Finally, the edges of the subset are doubled, creating a gap in the mesh.

Cutting tools that allow unnecessary new primitives usually include their creation for mesh quality and symmetry purposes. As described by Bielser and Gross (2000), the additional tetrahedral come from automatically inserting an extra node into a completely cut face. The addition of this node allows for a more symmetric subdivision and additional degrees of freedom in the cut surface. Adding these nodes to two adjacent faces, however, can cause the formation of intersecting tetrahedral. These tetrahedral are handled during re-meshing by incorporating additional tests for their intersection and choosing an alternative re-meshing template that would avoid self-intersection.

Schemes that take the third approach minimize the number of new primitives by creating new vertices only at the point of intersection between the tool and primitive. Since the intersection point can occur very close to the original vertices in the mesh, it is possible to create primitives having edges of widely varying lengths. Such primitives can be problematic for subsequent numerical calculations. To mitigate this problem, one can either shift the intersection location to the nearest vertex or modify these primitives after re-meshing. Despite the additional mesh-quality steps that might be required, minimal new primitive schemes have the advantage of reducing the overall number of primitives in the mesh after the cutting operation.

2.4.4 When Re-meshing is Performed

The cutting techniques that try to minimize the number of new primitives can be further categorized based on the time the re-meshing is performed. Re-meshing typically occurs:

(a) While the tool and the primitive are in intersection,
(b) After the tool is no longer intersecting the primitives, or
(c) After the tool has changed direction.

One can re-mesh a primitive along the cut path, as soon as the cutting instrument intersects it, as in progressive cutting; as soon as the instrument and primitive are no longer in collision, as with non-progressive cutting; or after the user has moved the tool out of the direction it was travelling in at the previous iteration.

Zhang *et al.* (2009) proposed a hybrid cutting method combining non-progressive cutting with progressive cutting, in which progressive cutting was applied on the outer hull while non-progressive cutting was applied in the inner core. It kept the visual reality while significantly increased the efficiency and stability for consequent soft-tissue simulation.

2.4.5 Representation of the Tool

The cutting techniques that simulate motions of the tool through the virtual environment and handle re-meshing can be further categorized based on the representation of the virtual tool. The tool is usually modelled as:

a) A single point of intersection;

b) An ideal object;
c) An object consisting of multiple primitives.

When modelling the tool as an ideal object, usually a single edge or triangle is used, allowing for simplified intersection tests.

When modelling the cutting tool as an object with multiple primitives, a central axis is usually used for intersection tests. This reduces the tool to a simplified model for computation, but retains the complex model for rendering. Ranal *et al.* (2007) proposed a method to simulate a cutting tool using an oriented cube that allows the simulation of tool thickness as well as performing the desired cutting operations on 3D models.

2.4.6 Section Summary

Just as the range of applications varies, the methods of cutting virtual objects can be very different. Many researchers have focused on developing cutting schemes for interactive surgical simulation requiring manipulations to be performed at haptic speeds. While it is required for some applications, other applications may find that less computationally intensive cutting techniques are more appropriate. In this thesis, as we want to build a realistic virtual surgical simulation system with real-time cutting simulation, bleeding effect and haptic feedback, we would take computational efficiency as a priority, as well as visual realism. We choose to move a virtual tool through the mesh as the definition of the cut path. Since cutting schemes are often linked with physically based models of the object being cut, there is a considerable penalty for meshes with unnecessary primitives, as the running time of dynamic solvers is on the order of the number of primitives in the mesh. Including more primitives than necessary can be costly, particularly when using solvers that are already prone to long running times, such as in the Finite Primitive Analysis. Therefore we would like to keep the number of new primitives at minimal. The element removal technique basically removes the elements that are intersected by the cutting tool. Despite its simplicity and computational efficiency, this method cannot present smooth cutting because of visual artefacts and the cut surfaces look unnaturally jagged. More appealing visual representations of incisions are made possible with mesh subdivision methods, but they often involve considerable increase of element count. Mesh adaptation can generate good cutting path without creating new elements, which fits our purpose very well. For a surgical training environment, the user expects immediate visual and haptic feedback while the cut is in progress, and a simulator that supports progressive cutting is needed. Hence we choose to develop a method of progressive cutting which splits triangles while following the motion of the cutting instrument. The instrument is represented by an object consisting of multiple primitives in the shape of a knife with the contact point on the tip of the blade, which could provide further realism.

2.5 Volume Deformation

Soft tissue simulation can be implemented utilizing either surface or volumetric models. Volumetric models such as the tetrahedral model are often chosen since they can simulate objects with an interior structure. However, topology modification of volumetric models is extremely complex and polygonal representations are usually obtained after a time consuming process of explicit segmentation and reconstruction. Surface mesh models are relatively easy to manipulate compared to volumetric models. Simple surface models however cannot display the object's interior structures. Bruyns *et al.* (2002) showed examples of cutting on surface model. Their single surface mesh looked like a hollow cover, which had no relation to the interior structure. Although complex surface models using multiple surface layers can offer thickness information, and can show cutting depth as well, it is inefficient to keep data for the hidden layers, which will not be displayed. In this thesis, a realistic and smooth progressive cutting is implemented on the outer surface model, and an algorithm is proposed to generate the groove and to reveal the interior structures around the cutting area by direct volume deformation and direct volume rendering. In fact, whenever the internal structure is important for the appearance or behaviour of a graphical object, a volumetric object representation is necessary. Direct volume rendering allows the efficient visualization of tomo-graphic 3D image data, using implicit segmentation based on transfer functions for colour and opacity values. Combining with direct volume deformation, it is capable of presenting a great deal of information about internal structures at the cutting site to enhance the fidelity and realism for the purpose of surgical training and planning.

Volume visualization is a method for extracting meaningful information from volumetric datasets through the use of interactive graphics and imaging. It is concerned with the representation, manipulation, and rendering of volumetric datasets. The objective is to provide mechanisms for peering inside volumetric datasets and for probing into voluminous and complex structures and dynamics. Volume visualization encompasses an array of techniques for projecting and shading a volumetric dataset, or properties thereof, and for interactively extracting from it meaningful information using transformations, cuts, segmentation, translucency control, measurements, and so forth. Typically, the volumetric dataset is represented as a 3D discrete regular grid of voxels (volume elements) and is commonly stored in a volume buffer (also called a cubic frame buffer), which is a large 3D array of voxels.

A voxel is the cubic unit of volume centred at the integral grid point. As a unit of the volume, the voxel is the 3D counterpart of the 2D pixel, which represents a unit of area. Thus, we can regard the volume buffer of voxels as the 3D counterpart of the 2D frame buffer of pixels. Each voxel has numeric values associated with it to represent some measureable properties or independent variables, for example, colour, opacity, density, material, coverage proportion, refractive index, velocity, strength and time of the real phenomenon or object residing in the unit volume represented by that voxel. The aggregate of voxels tessellating the volume buffer forms the volumetric dataset.

The source of volume data ranges from sampled data of real objects or phenomena, computed data produced by a computer simulation, or modelled data generated from geometric model. Examples of applications generating sampled data occur in medical imaging, such as computed tomography (CT), magnetic resonance imaging, and ultrasonography, biology (confocal microscopy), geoscience (seismic measurements), industry (industrial CT inspection), and molecular systems (electron density maps). Examples of application that generate computed datasets, typically by running a simulation on a supercomputer, occur in meteorology (storm prediction), computational fluid dynamics (water flow), and computational chemistry (new materials).

Considering physically based approaches for direct deformation and visualization of volume data, modern point based mesh free methods (Müller *et al.*, 2005; Rivers and James, 2007) seem to be the most natural approach to deal directly with medical volume data. Theoretically, no preprocessing is required and deformation could be directly performed on the volume if each voxel is modelled as a particle or phyxel. Although such an approach provides physically correct deformation, due to the computational complexity, to date it is not able to handle more than 100k elements at interactive frame rates (Müller and Chentanez, 2011). Besides, it is inconvenient to model real materials as it is based on geometry only (Faure *et al.*, 2011; Gilles *et al.*, 2011).

Most of the previous approaches addressing the deformation of volumes directly, i.e., without mesh extraction and/or simplification beforehand, are mainly based on space or ray deformation techniques: either a coarser structure, e.g. bounding boxes (Singh and Silver, 2004), a volume or asurface geometry (Masutani *et al.*, 2004; Xu *et al.*, 2011) is deformed. The deformation of the volume itself is computed as the displacement based on the deformation shape. This can be done either directly or indirectly by deformation of the viewing rays during rendering. However these approaches do not perform deformation at the finest level. To capture fine structures, extensive preprocessing (segmentation, geometric reconstruction) has to be performed.

Spatial Transfer Functions were introduced by Chen *et al.* (2003). They define a framework for specifying spatial transformation and deformation for volume objects. A spatial transfer function typically defines the geometrical transformations of every point in the volume. A backward-mapping operation must be performed (the inverse of the deforming function) to determine where to sample in the original volume dataset based on the current sample point on the ray. Although high quality rendering was attainable, due to the computational costs involved, this approach for volume manipulation is not yet able to support interactive manipulation on current desktop computers (Correa *et al.*, 2006) and it is categorized as a form of non-interactive manipulation (Nakao *et al.*, 2010).

Based on the linked volumetric representation, an approach to soft tissue deformation called 3D ChainMail is proposed by Gibson (1997), which is able to perform interactive deformation of volume datasets directly at the voxel level of the volume following the movement of individual voxel. The ChainMail algorithm itself is not a physically based deformation method but is capable of simulating material properties to some extent. Although it is originally only able to simulate plastic deformation, an additional relaxation step proposed by Gibson (1999) can be used to get more realistic and elastic deformations. Furthermore, the connected data structure allows easy manipulation and modification of the object topology, e.g., cutting and carving by removing elements or by breaking the links between elements. The algorithm is extended to model the differences between types of tissues and their interactions (Schill *et al.*, 1998). Li and Brodlie (2003) later developed the *Generalized ChainMail* algorithm to operate on arbitrary meshes in 3D. By extending the ChainMail algorithm and combines it with on-the-fly resampling and GPU ray-casting, interactive direct volume deformation and simultaneous visualization has been achieved for large volume datasets (Schulze *et al.*, 2007 and 2009). ChainMail algorithm has also been applied to 3D geologic surfaces (Faeth and Harding, 2009) for real time deformation. The latest implementations of ChainMail involve the simulation of needle insertion (Fortmeier *et al.*, 2012; Fortmeier *et al.*, 2013) and image-based palpation simulation (Fortmeier *et al.*, 2013). In our system, the cutting groove underneath the surface model is generated by applying the modified ChainMail algorithm on direct voxel level. Similar work has been done by Zhang *et al.* (2004) to model the internal structures of the cut object by appending new triangles to the surface mesh based on the position and degree of the scalpel penetration. However, by dealing directly with the volumetric data, our algorithm is capable of representing a great deal of information about the interior structures and material properties of heterogeneous tissues without any extra time-consuming preprocessing such as segmentation, simplifications, and adaptive hierarchy generation. In addition, our inner volumetric model can be easily extended to simulate operations such as drilling in a bone surgery or sculpting by simply removing the voxels in contact with the haptic device, which makes it a useful component to integrate into a comprehensive surgical simulation system.

2.6 Blood Simulation

Today’s surgical training systems commonly provide a powerful simulation of deformable anatomical models and surgical instruments. Realistic simulation of tissue cutting and bleeding is important components of a surgical simulator. Surgeons use a number of instruments to perform incision and dissection of tissues during surgery. For example, a coagulating hook is used to tear and spread the tissue that surrounds organs and scissors are used to dissect the cystic duct during cholecystectomy. During the execution of these procedures, bleeding may occur and blood flows over the tissue surfaces. For any virtual reality surgical simulator, bleeding has to be simulated in responsive to any actions the surgeon may be conducing in the virtual environment. Such simulation has to be performed in real-time, i.e. at frame-rate. However, the integration of interactive fluid models in a VR based surgical simulator still remains a very challenging problem to date.

Most surgery procedures involve some kind of fluid which could add substantial realism to a variety of application scenarios. Furthermore, the fluid can be employed to improve treatment planning, e.g. in aneurysm surgery. The example of fluid providing a realistic visual cue could be found in many open surgeries. Unpredictable blood splashing might be resulted from incisions on organs. The existence of fluid can sometimes obscure a surgeon’s view and make the operation more difficult. Saline fluid is flooded into the operation area for cleaning purpose. The mixed fluid of blood and water are commonly sucked away or drained from the operation area from time to time. Also in endoscopic training systems, the simulation of blood flow due to injured arterial vessels could significantly improve the visual feedback. In aneurysm surgery, fluid simulation can provide useful information on aneurysm hemodynamics and pathology. This information can be utilized as a design criterion for stents which are used in the treatment of aneurysms. Thus, fluid simulation can be used to improve the quality of the treatment planning and decrease the surgical risk. It is clear that handling fluid is an essential part of a surgery. Fluid dynamic modelling can therefore enhance the realism in surgical simulation. Without fluid, surgical simulations appear dry and clean, which would be unrealistic for most surgical procedures. Even when the inclusion of fluid dynamic might not have much effect to haptic feedback, the visual realism provided by simulation of fluid deems it necessary for surgical simulations.

The computer graphics community has developed strikingly realistic techniques of animating fluid flows. One of the most notable efforts was by Stam and Fiume (1993), simulating smoke, steam or liquid by solving the Naiver-Stokes equation within a 3D grid. Another approach by Fedkiw *et al.* (2001) used vorticity confinement to capture the small features of the fluid and rendered the smoke with a photon-mapping technique for higher realism. In the work by Kim *et al.* (2008), simulation of fluids at multiple levels of detail was achieved using wavelets. However, none of these methods is suitable for real-time environments, as they are intended for animations in which each frame is computed over many hours using high-performance computers.

Real-time techniques for the generation of fluid may be categorized into two major groups: physical and non physical. The physical approaches (Harris, 2008; Kerwin *et al.*, 2009) offer the most accurate simulation of the underlying fluid dynamics. In general, these approaches can be further categorized into two classes: Lagrangian based methods and Eulerian based methods. Both methods are based on the Navier-Stokes Equations, which are the fundamental equations governing the motion of a fluid. In the Lagrangian viewpoint, the fluid is simulated as discrete blobs of fluid. Each particle has various properties, such as mass, velocity, etc. The benefit of this approach is that conservation of mass comes easily for irregular discretization. Monaghan (1992) introduced the smoothed particle hydrodynamics (SPH) method into the computer graphics community to address the irregular discretization issue in lagrangian method .It defines a smoothing kernel to interpolate the physical properties (velocities, densities, etc.) at an arbitrary position from the neighbouring particles. SPH is now becoming a more and more popular technique in the field of fluid simulation (Zhang *et al.*, 2008; Solenthaler and Pajarola, 2009; Krog and Elster, 2010; Zhang *et al.*, 2011). The Eulerian viewpoint, on the other hand tracks fixed points inside the fluid. At each fixed point, quantities are traced such as the velocity of the fluid as it flows by, or the density of the fluid as it passes by. The Eulerian approach corresponds to grid based techniques. Grid based techniques have the advantage of having higher numerical accuracy, since it is easier to work with spatial derivatives on a fixed grid, as opposed to an unstructured cloud of particles. However, grid based techniques often suffer from mass loss, and are often slower than particle based simulations. Finally, grid based simulations often do much better tracking smooth water surfaces, while particle based approaches often have issues with these smooth surfaces. A bunch of other works based on Eulerian method (Kim *et al.*, 2007; Brochu and Bridson, 2009; Batty *et al.*, 2010; Chentanez and Müller, 2011) have also produced very realistic result of fluid phenomena. These physical based methods are, however, computationally very demanding and the cost increases dramatically with increase in the grid size or the number of particles. Besides, the additional burden for collision detection/response during the tool tissue interactions require large amounts of computations and can cause performance bottlenecks. Although simulations with GPU allow for computations to be performed independent of the CPU, the CPU–GPU data transfer limitations may restrict the maximum number of particles or grid size in a simulation, thereby affecting the realism. In addition, time step and numerical errors may pollute the solution.

The non-physical approaches, however, are based on rendering techniques, where the focus is to create visually appealing rather than physically accurate results. These include techniques such as dynamic-textures (Daenzer *et al.*, 2007), 3D textures (Kühnapfel *et al.*, 2000), gravity maps and height maps (Phenomena, 2004; Halic *et al.*, 2010).

The need to incorporate the effects of bleeding in VR-based surgical simulators has been well recognized in previously developed simulators. In a cricothyroidotomy simulator developed by Bhasin *et al.* (2005), bleeding simulation was implemented to add realism when cutting tissues with a scalpel. Although their method had a visually appealing result, it requires the bleeding image to be updated in every CPU cycle, which could be prohibitive for more complex procedural simulations. A similar approach for simulating bleeding in a laparoscopic surgery of the liver was used in the work by Neyret *et al.* (2002). The visual realism was not enhance compared to previous work but they introduced more control, such as gravity and surface friction for blood flow; however, the authors noted performance bottlenecks in rendering complex bleeding scenarios. The work by Sweet *et al.* (2002) focused on an image-based approach to simulate blood flow in a virtual prostate surgery simulator. Bleeding during the procedure was simulated with pre-recorded movies of the diffusion of a red dye in the fluid, superimposed onto the camera view of the simulator. Although this provided realism to the simulator, this technique cannot be generalized to all types of bleeding experienced in a surgery. Qin *et al.* (2007) developed a fast technique to simulate blood stream and blood flow for an orthopaedic surgery simulator. They simulated blood flow from the vessels during the incision using a specialized physics processing unit (PPU) hardware and a GPU for rendering bleeding surfaces. However, this specialized hardware-based method did not scale well, due to the limited data transfer rates. Moreover, the end of support for specialized PPU hardware limited the applicability of this technique. In a virtual hysteroscopy surgery simulator (Zátonyi *et al.*, 2005), blood flow in a uterine cavity was simulated in 2D. Even though the bleeding simulation was realistic, the 2D approach was limited to a fluid environment. In a temporal bone dissection surgery simulator (Kerwin *et al.*, 2009), both bleeding and irrigation with water was simulated in 2D using a GPU-based technique. The simulator demonstrated satisfactory results from a user study. The major drawback of this method is that it cannot simulate 3D effects. A particle based solution to simulate the interactions between blood flow and vessel wall for virtual surgery is proposed by Qin *et al.* (2010). Experiment results demonstrate that the proposed method has potential to provide real time and realistic interactions for virtual vascular surgery systems. However, the authors also observed that when Marching Cubes based rendering method is employed, the performance greatly deceases. In the work by Halic *et al.* (2010), an effective GPU-based technique for both bleeding and smoke simulation was presented. Computationally expensive physical simulations of bleeding and smoke were avoided by using image-based techniques. Both the bleeding and the smoke techniques utilized GPU, thereby eliminating overloading of the CPU. The major problem of this technique is the bottleneck problems between CPU and GPU data transfer.

Given that the major goal of bleeding simulation in virtual simulators is to create a realistic visual effect without introducing significant additional load on the CPU, it is apparent that non-physical methods are more suitable for present day surgical simulators, as many of the physical approaches require too much computation to be integrated into real-time applications. This is critical for current surgical simulation systems because of the enormous computational load already placed on the CPU from simulating deformations, collisions and response characteristics.

2.7 Haptic Interaction

Of the human sensorial modalities (visual, auditory, touch, smell and taste), two main modalities are currently used in surgical simulation: visual and touch. Smell and taste will be included in the future with new products such as the ScentPalette from EnviroScent (Ball Ground, GA, USA). Auditory cues are sometimes used to alert a user to a fault or for guidance and can be important for the correct learning of certain procedures using high-speed power tools, such as burr-based bone and tooth drilling. The visualization component is provided using either two-dimensional or three-dimensional displays and is well documented. The sense of touch has a great role in enhancing a user's perceived fidelity of computer simulations for medical procedures.

Haptics solutions are less mature than visual display technologies. In particular, haptics require bidirectional input and output, which is difficult to model accurately due to the large number of the different touch receptors involved. Haptics can be considered in two main categories: tactile feedback and force/torque feedback. Tactile feedback is sensed by receptors in and just under the skins surface allowing humans to detect if a surface is smooth or rough, hot or cold, as well as conveying pain. Force/torque feedback resists motion and/or rotation, for instance, stopping a person's hand falling through a table top as they touch it. The biological receptors providing this feedback are in muscles and at joints allowing a person to know where their hand is in space, even with closed eyes (proprioception). Both tactile and force feedback can be crucial to the success of carrying out a medical procedure.

The term "force feedback" is often used in place of "haptic feedback". However, these terms are not interchangeable. In a general case of proprioceptive feedback, where a person interacts with a simulated scene, both forces as well as torques must be experienced. This requires six degrees of force (DOF) feedback but is not typically provided because of the higher cost of manufacturing devices that can provide torque as well as directional force feedback.

Several commercial devices usable for surgical simulation have become available. One of the most commonly used devices is the PHANToM from Sensable Technology^{TM}, which is now available in several models. PHANToM is an acronym for Personal HAptic Interface Mechanism and evolved from haptic research at the MIT Artificial Intelligence Laboratory. Another popular force feedback device is the Laparoscopic Impulse Engine from Immersion Corporation, which also enables surgical tools to be tracked and manipulated in 3D space. The device interfaces with a computer via a PVI card and the development kit supports Windows-based and Silicon Graphics computers. The Mediseus Epidural simulator (Medic Vision) is a commercial example of a needle insertion simulation using force feedback. The simulation can be run from a laptop. It gives a vocal response if the user makes mistakes and produces an objective report for the student (Mayooran *et al.*, 2006). A relatively low-cost SensAble PHANTOM Omni is encased inside the system using a modified syringe end effector at a fixed insertion point. This transforms the three positional DOF to one positional and two orientation DOF.

A key issue in integrating force feedback devices into a surgical simulation system is the update rate required for high fidelity. Although the visual display can be updated at 30 Hz, the haptic interface update rate should be around 1,000 Hz for stability reasons and to obtain a responsive interface. This can be accomplished by a multi-rate simulation with a high-bandwidth force feedback loop as described by Cavusoglu and Tendick (2000).

2.8 Chapter Summary

The chapter has presented a review of existing work of six main components of a surgical simulator that is relevant to this thesis: data sources and pre-processing, soft tissue modelling, surface cutting, volume deformation, fluid simulation and haptic interactions.

Surgical training has traditionally been one of apprenticeships, where the surgical trainee learns to perform the surgery under the supervision of a trained surgeon. This is costly, time consuming, and is of variable effectiveness. Training using virtual reality simulator is a low cost, low risk, and hence a viable option to supplement standard training.

In this thesis we focus on a computer-assisted approach based on virtual reality techniques using 3D patient dataset. Although numerous methods have been established to construct surface meshes from CT and MRI dataset, direct volume rendering is still superior in presenting interior structures. In order to perform virtual reality based pre-operative rehearsal in the field of open surgery, the system has to support both interactive soft tissue cutting and accurate deformation with virtual organs. Also, fluid simulation is an indispensable part of surgical simulation system as it is a valuable visual clue for the users. This thesis presents a new integrated simulation framework and aims to develop a training simulator for surgical cutting and bleeding in a general surgery.

Chapter 3 Real-time Hybrid Surgery Cutting

3.1 Introduction

Virtual surgical simulation is a technology dedicated to medical training and surgery planning. To achieve a high degree of realism, the virtual surgical simulation systems need to support the following: (1) heterogeneous scenes composed of different states of matter (solid, liquids, and gases); (2) complex geometry and material properties of objects within the scene; (3) dynamic and real-time interaction (palpation, cutting, etc.) between virtual objects and tools manipulated by the user; and (4) multimodal (visual, auditory, and haptic) rendering of the results to the user. The complexity resulting from the four requirements above necessitates the dedicated techniques of modelling, simulation, and rendering. In this thesis we aim to discuss some of these key techniques and to design and develop a prototype of surgery simulation system.

Cutting simulation is the key component in surgical simulation as cutting is a common operation in both conventional and minimal invasive surgeries. Most surgical tasks begin with an incision to expose the surgical region. Despite advancements in computational biomechanics, modelling and simulation of soft tissue cutting still remain one of the most challenging problems in surgery simulation. The difficulties lie with the need to model surgical cutting and the nonlinear geometry and material behaviour exhibited by soft organs (Wittek *et al.*, 2008), and to achieve high computational speeds for real time interaction.

Soft tissue simulation can be implemented utilizing either surface or volumetric models. Surface models are relatively easy to manipulate and surface-based cutting methods can provide smooth and realistic cutting effect at a low computation cost, but they suffer from important shortcomings. They cannot display the object's interior structures to show the result underneath a cut. With just a one-layer surface model, the object after cutting looks like a hollow cover and could not represent the different material properties, as the examples shown by Pan *et al.* (2011). Although layered surfaces can be employed, a surface mesh is generally unable to simulate progressive cutting in depth. Volumetric models, on the other hand, can incorporate appropriate internal structures and material properties for situations in which high-fidelity virtual environments are required. In fact, whenever the internal structure is important for the appearance or behaviour of a graphical object, a volume object representation is necessary.

In the current surgical simulation systems, it is generally not possible to provide a physically-based simulation of the internal forces involved in the deformation, due to the huge computational cost required. Instead, it is considered more important to provide realistic visualizations of the internal structures at the cutting site as the visual clue, especially for surgical training and planning purposes (Lin *et al.*, 2007). Therefore, priority should be given to the realistic surface deformation and smooth cutting before there is an opening, and subsequently the users often focus on the exploration of the interior features. We hence propose a hybrid method to deal with the cutting of heterogeneous objects, which consists of an outer surface model and an inner volumetric model. A node-snapping technique is presented to modify the surface topology of the objects, without adding new elements. Progressive smooth cut is generated by duplicating and displacing mass points that have been snapped along the cutting path. Node-snapping technique is modified to allow haptic-controlled deformation of the incision, i.e., the crack could be pulled or manipulated for further operation. Our algorithm is easily extended to simulate other surgical operations such as suturing by knotting the two sides of the incision back together. A volumetric deformable model is employed underneath the surface, with considerations on the internal structures and material properties of heterogeneous objects without extra time-consuming preprocessings such as segmentation, simplifications, and adaptive hierarchy generation. A cutting gutter is generated by the modified 3D ChainMail deformation applied on the underneath volumetric model. In addition, our inner volumetric model can be easily extended to simulate other operations such as drilling in a bone surgery or sculpting by simply removing voxels in contact with the haptic device, which makes it a useful component to integrate into a comprehensive surgical simulation system. We build a tight connection between these two data models so that the volumetric gutter is formed according to the outer surface cut openings progressively as a hybrid model. In the next section, we will focus on the manipulation of the outer surface model.

3.2 Surface Manipulation

3.2.1 Surface Deformation

When a tool collides with a deformable object in a cutting procedure, the object's surface deforms until the applied force becomes larger than the yield limit of the material being simulated. The deformation is determined by the contact point and the motion direction of the tool. Various techniques can be found in literature for the simulation of deformable objects. These techniques can be categorized into two main approaches: geometrically based approaches and physically based approaches. In geometry-based techniques, such as Bezier/B-spline based procedures and freeform deformation techniques, the object or surrounding space is deformed by manipulating vertices or control points. These techniques are relatively faster and easier to implement. However, they do not capture the physics of the deformation.

The physics-based techniques, on the other hand, aim to model the physics involved in the motion and interactions. One of the simplest physics-based models, and thus the most likely to achieve real-time interactivity, is the mass-spring elastic network. Mass-spring systems consist of a set of point masses, connected to each other through a network of springs and dampers, moving under the influence of internal and external forces. This technique has been used extensively by computer graphics researchers in simulating soft tissue and cloth behaviour. However, for the purpose of deformation modelling, the mass-spring model suffers from many disadvantages. It is difficult, and sometime impossible, to determine the parameters of hundreds of thousands of springs, dampers, and masses to represent the global behaviour of heterogeneous soft tissues especially if nonlinear and/or viscoelastic behaviour is to be captured. It is difficult to enforce global properties like incompressibility when using such models and the problem is exacerbated when one tries to use relatively fewer particles to reduce computational time. Relatively stiff springs are necessary to model hard tissues, jeopardizing the stability of the solution scheme, and requiring the numerical temporal integrator to take minute time steps. Finally, anisotropic distribution of mass points necessitates fine-tuning for individual organ geometry, difficult in controlling the variation of forces and deformations across the geometry as well as integrating tissue properties into the model.

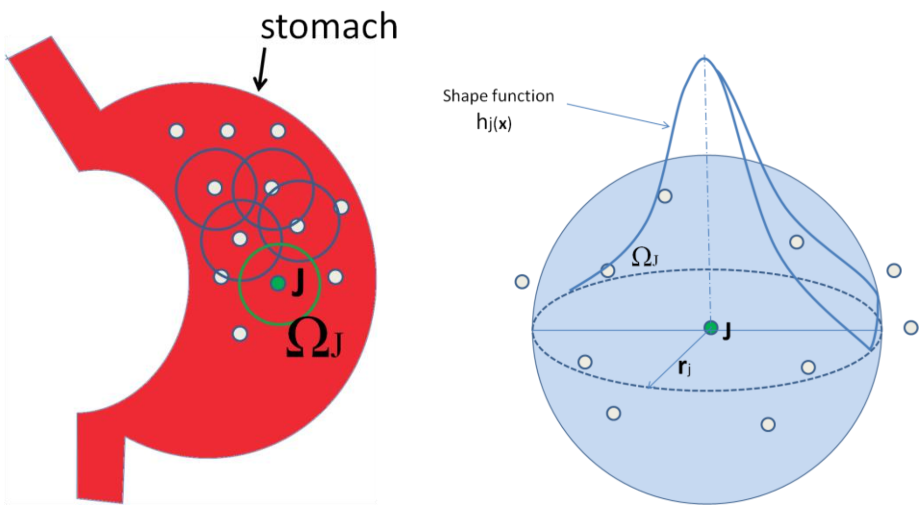
As an alternative to mass-spring models, the use of more robust but expensive finite element analysis procedures has been proposed. In spite of accuracy and robustness, finite element techniques suffer from certain drawbacks in real time simulation. Firstly, the need for numerical integration and volumetric meshing results in a slower-than-real-time performance unless extensive precomputations are performed. Furthermore, the contact between tool and tissue must occur only at nodal points. Hence, for a smooth visual and haptic display, a very fine mesh needs to be utilized, resulting in extensive memory usage and high computational overhead. Large deformations and nonlinear response of tissues cause the finite elements to behave badly or totally fail unless re-meshing is performed frequently. Meanwhile, change of topology, for example, during the simulation of surgical cutting, necessitates re-meshing, which destroys any precomputed data. The number of computations would be increased on the fly, resulting in seriously degraded real time performance.

There are, however, positive features of both the mass-spring and finite element analysis schemes. Mass-spring models offer simplicity and speed and do not suffer from the deleterious effects of mesh distortion. Finite element schemes are used to solve the partial differential equations that govern the deformation and motion of soft tissues and only a limited number of empirically determined parameters are necessary.

Hence, an ideal combination of mass-spring and finite element-based techniques is desirable. Such an “ideal” scheme should solve the governing partial differential equations, but not suffer from any of the problems associated with a mesh (e.g., mesh distortion and re-meshing when large deformations or surgical cutting have to be modelled). It should be very flexible, in the sense that it should allow arbitrary local refinement and multiresolution capability to zoom into region of “action” without having to unduly refine the discretization over the entire computational domain. Finally, the ideal scheme should allow the simulation of matter, irrespective of its state of solid, liquid, or gas, within a single computational framework. This is essential since one needs to simulate dissection, bleeding, and, possibly, smoke generation within the same scenario without having to switching between various modelling schemes stitched together through tenuous non-physical links.

Point-associated finite field (PAFF) is such a scheme for deformation. It is based upon the equations of motion dictated by physics without jeopardizing the computations. This method resides between the continuum mechanics and particle-based approaches. Like mass spring models, it is a point-based approach, using only the nodes of a 3D object for the displacement and force calculations. PAFF approximates the displacement field by using nonzero functions over small spherical neighbourhoods of nodes. Like FEM, this technique uses a Galerkin formulation to generate the discretized versions of the partial differential equations governing the deformable object's behaviours. PAFF supports simulation of large deformations as well as topology modifications such as cutting. It can also be used to simulate other procedures involving particles, such as smoke generation during cauterization. Although the technique’s brute-force implementation is computationally intensive, users can generate localized solutions in real time. In our application, the surface deformation of the soft tissue is computed using the mesh-free physics-based PAFF technique up to the point of tissue rupture.

In PAFF, an object is represented, irrespective of its state, as a collection of particles or nodes that serves as the computational primitives (Figure 3.1), much like the mass-spring models. But unlike mass-spring models in which the governing partial differential equations are solved, the method is “meshless”, since no direct link exists between the computational particles. The particles pose a finite “region of influence” which smears out their effects and coordinates their motions during simulation. Since no mesh is used, none of the problems associated with a mesh are encountered. In our surgical simulation system, it may be assumed that the deformation zone is localized within a “region of influence” of the surgical tool tip. The influence zones are spherical in shape and determined by a function known as the shape function or approximation function.



(a) (b)

Figure 3.1: The point-associated finite field (PAFF) technique. (a) A general three-dimensional body (e.g., stomach) discretized using a set of nodes. Each node has an associated spherical influence zone. (b) The approximation function $h_J(x)$ at node J is “bell shaped” and is nonzero only on the spherical influence zone of radius r_J centered at node J.

The PAFF idea is quite simple. One assumes that the variable of interest (the displacements in the three Cartesian directions, in this case), say $\boldsymbol{u}$ at any point $\mathbf{x}$ inside the computational domain, can be approximated using the following formula

$$u_h(x) \approx \sum_{J=1}^{N} h_J(x)\alpha_J \tag{3.1}$$

where N is the total number of particles in the domain, and $h_J(x)$ is a shape function of the form

$$h_J(x) = W_J(x)P^T(x)A^{-1}(x)P(x_J) \quad J = 1,\ldots N \tag{3.2}$$

with

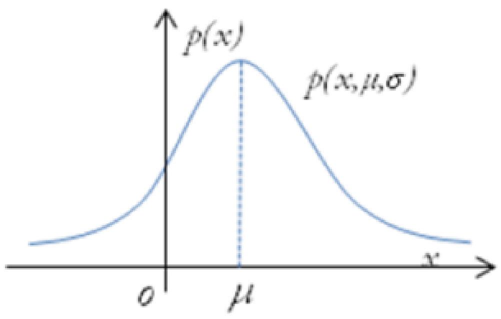
$$A(x) = \sum_{I=1}^{N} W_1(x)P(x_1)P^T(x_1) \tag{3.3}$$

$W_J(x)$ is a radial weighting function which is nonzero only on the spherical influence zone centred at node J. To ensure that the approximation in Equation (3.1) is globally continuous, $W_J(x)$ needs to be chosen such that it has a zero slope on the surface of the sphere as well at its centre. For those familiar with the theory of wavelets, this is much like the condition of vanishing moments. Approximation spaces with a very high degree of smoothness can easily be generated by choosing appropriate weight functions.

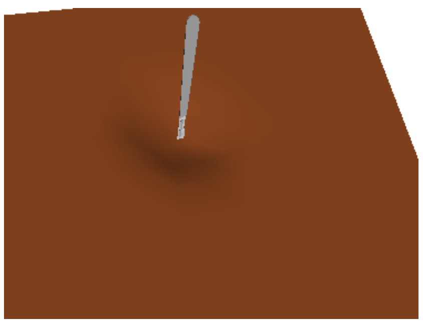
The vector $P(x)$ contains polynomials ensuring consistency up to a desired order (in our implementation we have chosen $P(x) = [1, x, y, z]^T$ to ensure a first order accurate scheme in 3D, similar to bilinear finite elements). In all these expressions, N is the total number of particles in the domain and α_J is the nodal unknown at node J.

In PAFF, the approximation in Equation (3.1) is used to solve the governing partial differential equations of motion of a continuum, using a method known as point collocation. In this technique, the partial differential equations, as well as the boundary conditions, are satisfied at the nodal points. Compared to the finite element technique, this method is vastly simplified, since no computationally intensive numerical integration is used.

A straightforward implementation of PAFF will not achieve real-time performance. A key assumption is hence made such that the surgical tool - soft tissue interaction is local and the deformation field degrades rapidly with increase in distance from the surgical tool tip. This localization assumption is based on a Gaussian function in our application. In Figure 3.2(a), the x axis represents the distance of a particle to the contact point between the surgical tool tip and the surface, and the y axis represents the depth of the deformation along the direction of penetration. It can be clearly seen from Figure 3.2(b) that, the further the particle is from the contact point, the less displacement it experiences.



(a)



(b)

Figure 3.2: (a) Shape function for our application. (b) Surface deformation.

3.2.2 Surface Mesh Cutting

When the deformation of the surface reaches it yield limit, a rupture or surface cut will occur. As for cutting models, there are mainly two groups, including element removal and re-meshing (Figure 3.3). The element removal method directly removes elements intersecting the cutting tool (Cotin *et al.*, 2000; Choi *et al.*, 2009). This method is computationally efficient. However, it violates the physical principle of mass conservation due to the removal of elements and it is often difficult to present a smooth cut. The element re-meshing method recreates the cutting and forms a cut opening in the mesh (Bruyns *et al.*, 2002; Sela *et al.*, 2004; Sela *et al.*, 2007). It can be further categorized as mesh subdivision and mesh adaption methods.

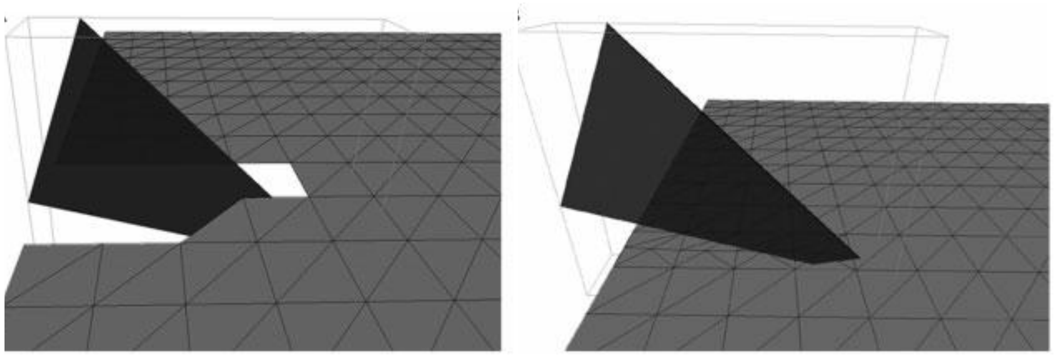


Figure 3.3: Handling of surface cutting: (a) Removal of intersected primitives. (b) Re-meshing the intersected primitives.

The mesh subdivision method divides elements along the cutting plane to generate a relatively smooth cutting path (Bielser *et al.*, 2004; Huy *et al.*, 2006). The deficiency of this method is the increment of the number of elements and the creation of smaller or degenerated elements. The newly increased elements may slow down the computation and the degenerated elements as well as the edges with widely varying lengths may cause the instability of numerical calculations of deformation.

The mesh adaption method, also known as node snapping, generates the cutting path by aligning the nearby mass points and edges on it. These edges are duplicated and separated apart to form the cut opening. In this process, no new facet is created and the mass is conserved. As stated in Nienhuys and van der Stappen (2001), degeneracy issues may be easily raised due to the aligned edges and the incision simulation lags behind the scalpel. The progressive cutting simulation (Kim *et al.*, 2010; Turkiyyah *et al.*, 2011) is required to generate the cut opening closely following the movement of the scalpel. We present an efficient cutting algorithm based on the idea of node-snapping that is capable of visually simulating progressive cutting with minimum increase in the number of new elements.

Node-snapping methods have been used in geographic information systems for nodal coordinate adjustments of digitized data, and in CAD tools for identification of features and cleanup of data with node merging and weeding. Nienhuys and van der Stappen (2001) proposed a similar algorithm for the modification of object geometry. Later Lin *et al.* (2007, 2008) applied the approach for progressive cutting.

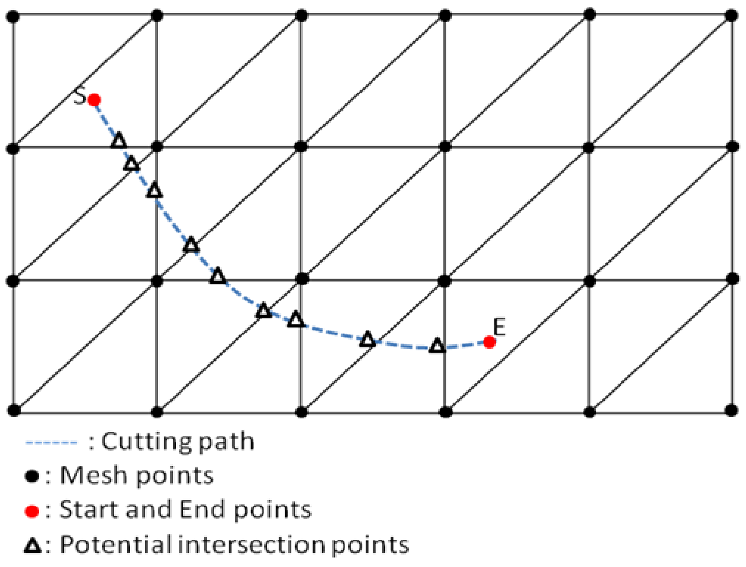


Figure 3.4: Cutting path intersects with mesh points.

The node-snapping method starts with collision detection. After the initial collision of the haptic tool with the object surface, intersection points between the cutting path and the underlying polygon edges are calculated as shown in Figure 3.4. It is worth mentioning that the marked intersection points are only potential as the cutting is a progressive process. Whenever an intersection point is calculated, the local area of the mesh is updated before move to the next intersection point. Figure 3.5 illustrates the details of the process. As the cut progresses, the mesh point nearest to the intersection point is snapped to the cutting path and the cut will be generated by dividing the mesh model along the cutting path and temporarily ends here. This is especially important for visual realism when the cutting tool moves very slowly and the cut should still be updated in real time. The re-oriented polygon edges continuously follow the cutting path.

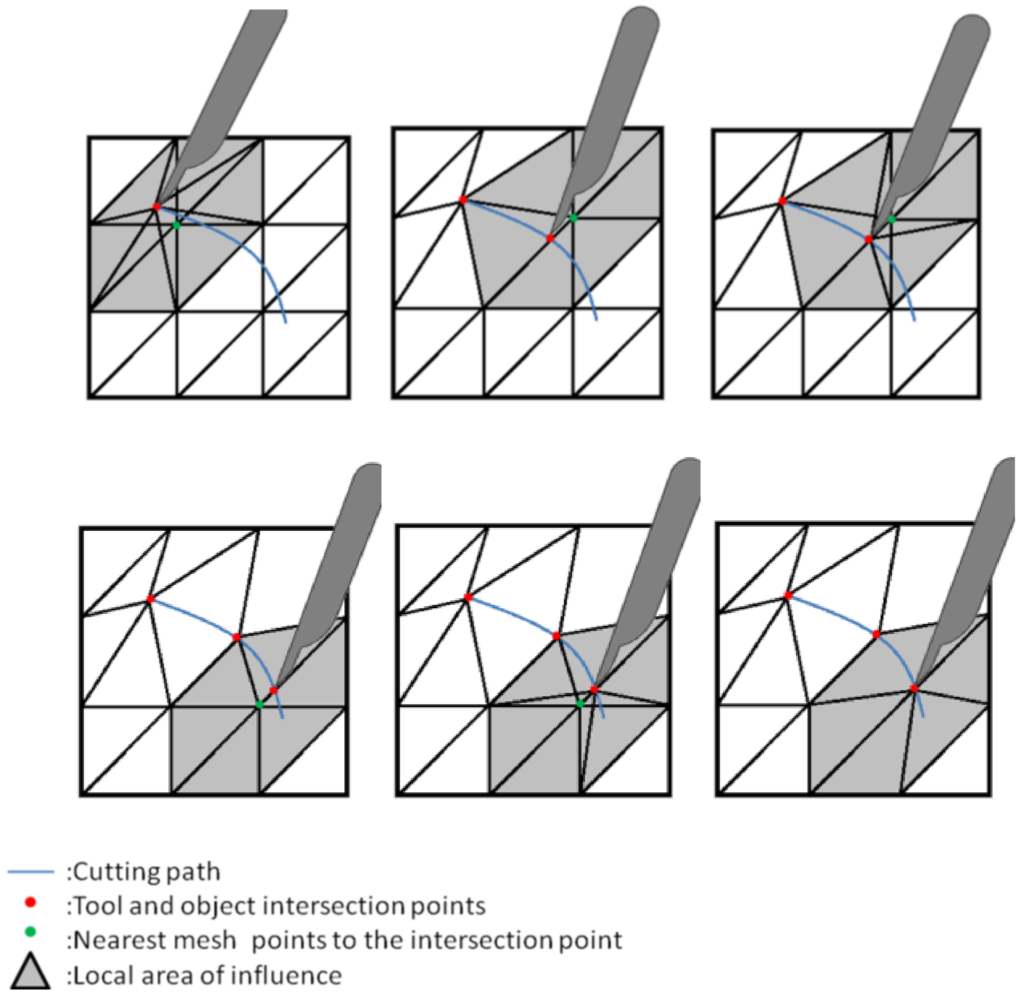

:Cutting path
:Tool and object intersection points
:Nearest mesh points to the intersection point
:Local area of influence

Figure 3.5: The progressive cutting.

For each intersection point, after the node-snapping, the next task is to open up the cut by duplicating and displacing vertexes that have been snapped along the cutting path. The snapped points except the starting (*S*) and ending points (*E*) of the cut are duplicated and directly displaced towards the two sides of the cutting path, such as the vertices S_{i1} and S_{i2} for the original vertex S_i in Figure 3.6. The displacement direction is perpendicular to the cutting path. The original snapped points such as point S_i are then deleted. All polygon edges connected to the deleted points are reconnected to their duplicated points. As shown in Figure 3.6, edges originally connected to point S_i are now connected to point S_{i1} or S_{i2}, depending on which side of the cut they are located at. In particular, both S_{i1} and S_{i2} are set to be connected to the starting cut point *S*. After some points on the surface are snapped, duplicated and deleted, surface triangles in the vicinity of the cut need to be updated according to the reconnected edges near the cut.

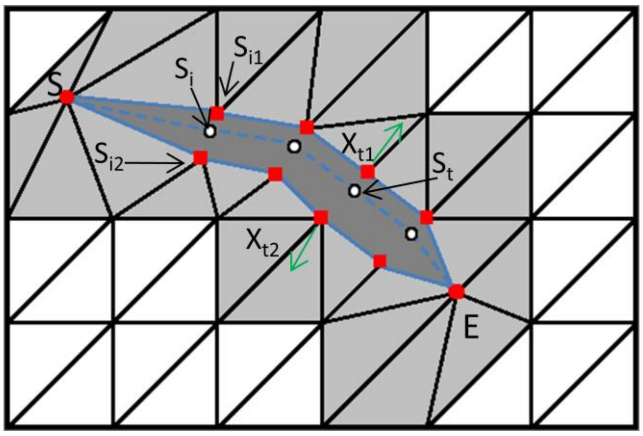


Figure 3.6: Cut opening generation.

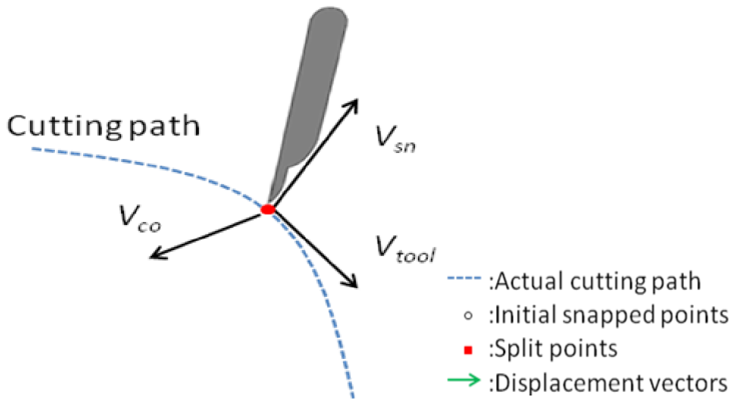


Figure 3.7: The definition of the cut opening vector.

The displacement vectors X_{t1} and X_{t2} in Figure 3.6 are defined on the tangent plane of the original vertex. The normal of the tangent plane of a vertex is calculated as the average normal of its neighbouring triangles. Let W be the cut opening width at point i, which is currently set as a user-defined controlling parameter representing the tension of the simulated tissue, the displacement vectors of the duplicated points can be calculated as:

$$X_{t1} = +\frac{W}{2}V_{CO} \tag{3.4}$$

$$X_{t2} = -\frac{W}{2}V_{CO} \tag{3.5}$$

The new positions of the duplicated mesh points can hence be calculated as:

$$S_{t1} = S_t + \frac{W}{2} V_{CO} \tag{3.6}$$

$$S_{t2} = S_t - \frac{W}{2} V_{CO} \tag{3.7}$$

where V_{CO} is the unit vector along the displacement direction which can be calculated according to Figure 3.7 as :

$$V_{CO} = \frac{1}{|V_{tool} \times V_{sn}|} V_{tool} \times V_{sn} \tag{3.8}$$

where V_{tool} and V_{sn} are the unit vectors along the directions of the tool travelling and the tangent plane normal at node S_t, respectively.

Although this cutting method is not physically based, the generated cut is unconditionally smooth with high level of realism. The method is computational more efficient than the physically based numerical integration methods where the cut is generated by the spring forces of disconnected springs. By duplicating and displacing mass points that have been snapped along the cutting path, node-snapping algorithm could generate very smooth cut without adding new elements. Figure 3.8 shows the cutting results on the surface on different resolution. New edges are generated and connected to the neighbouring mesh points. Figure 3.8 (b) also shows an example of multiple cuts on the same mesh. Cuts with different cut opening width *W* are also presented in Figure 3.9.

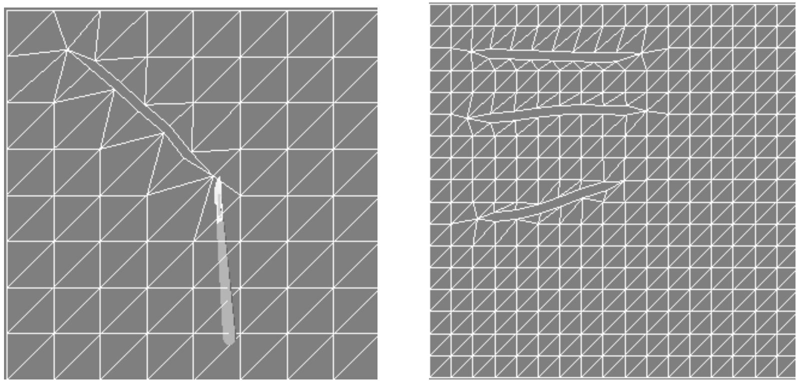

(a) (b)

Figure 3.8: Surface cut on the mesh plane.

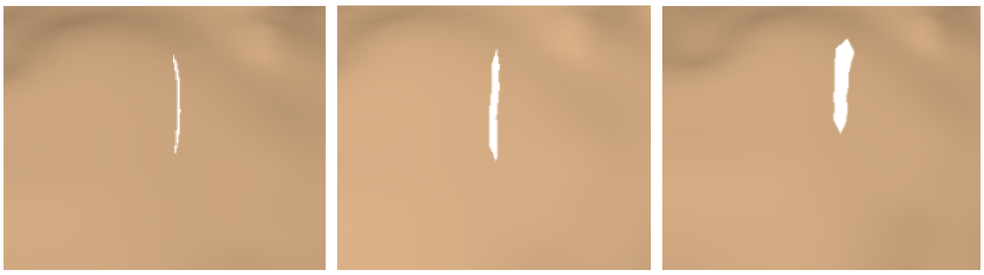


Figure 3.9: Cuts with different cut opening widths *W*.

3.2.3 Wrinkle Effect

During the progressive cutting, collision between the blade and the geometry has been constantly detected. Once there is no more collision detected, which means that a cutting operation has finished, the common next step in a surgery is to pull open the cut to expose the underline surgical region. Due to the elasticity of the soft tissue, the wrinkle effect will be generated along the cut path on the surface during the pulling.

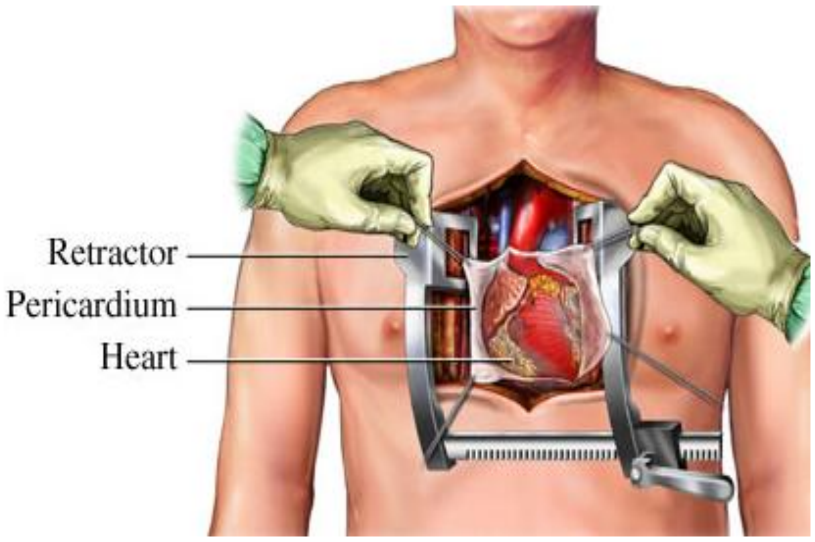


Figure 3.10: The use of retractor in a real surgery-medical illustration. (http://catalog.nucleusinc.com)

Retractors or spatulas are used in surgery to separate the cut surfaces while the surgeon attempts to make further cuts or to focus on the underline region of interest (Figure 3.10). In a real surgery, when a surgeon uses a retractor to hold back soft tissues such as skin to open up the cut, there will be fold along the cutting edge, because of the surface elasticity. Wang *et al.* (2006) used two hand-held haptic devices to simulate the contact with retractors. However, they did not implement the wrinkle effect when the cutting edges are pulled away by the retractor. In addition, complex calculation is required to obtain the intersections between the retractor polygon and the surface object. Correa and Silver (2007) have implemented a very realist wrinkle effect along the cut (Figure 3.11), but the deformation is a part of the rendering pipeline, which differs from traditional deformation methods where deformation is considered as a modelling problem and a new mesh is required. Obviously, such an implementation does not fit our purpose. We design a decay function to displace the mesh points along the cutting path to simulate the wrinkle effect. The concept is similar to the surface deformation before rupture, except that Gaussian function is replaced by the decay function. In Figure 3.12, the *x* axis represents the distance of a mesh point to the cutting path, and the *y* axis represents the depth of the deformation along the direction of the penetration depth. It can be clearly seen that, the deformation field degrades with increase in distance from the surgical tool tip in a damping way to form the wrinkle effect. Even though the concept of our method is very simple and straightforward, it can produce a very realistic and smooth wrinkle effect without jeopardizing computations. Figure 3.13 is an example of the wrinkle effect applied on a surface mesh.

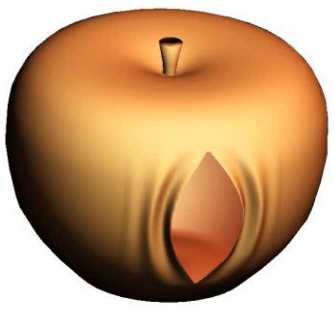


Figure 3.11: Rendering of wrinkle from the work of Correa and Silver (2007).

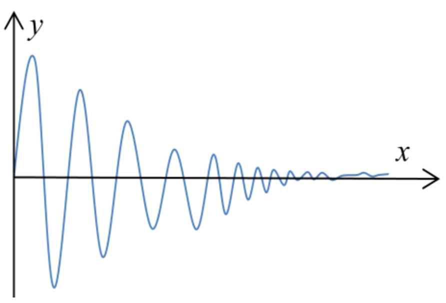


Figure 3.12: Decay function.

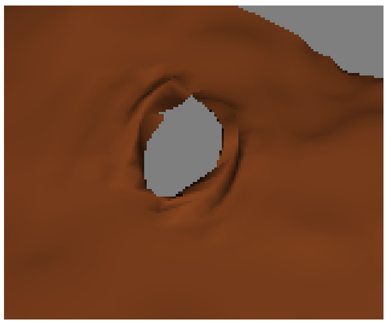


Figure 3.13: Wrinkle effect.

3.3 Volume Deformation

As discussed previously, cutting algorithms based on surface mesh models are easy and simple to manipulate. However a surface mesh is only suitable for modelling cuttings on membrane-like structures, such as intestine or gall bladder, with a great deal of information about the interior structures and the material properties of the heterogeneous tissues disregarded in most surface cutting algorithms.

In contrary, volumetric models provide a good means to represent interior structures and mechanical properties of heterogeneous volumes. By dealing directly with the volumetric data, the heterogeneous structures could be achieved without extra preprocessing. In fact, whenever the internal structure is important for the appearance or behaviour of a graphical object, a volumetric object representation is necessary. In this section we propose novel algorithms to realistically simulate surgical cutting on deformable anatomy objects consisting of an inner volumetric model in addition to an outer surface model.

As discussed in Chapter 2, our cutting groove on the underneath volumetric model is implemented by modified ChainMail algorithm. The underlying data structure of the ChainMail algorithm can be compared to a chain mail in the 2D case extended by a third dimension in the 3D case. It is the key to the entire algorithm. The basic idea of the algorithm works as follows. The elements of an object are linked together like elements of a chain mail. If one element is moved there is a chance that it will also make its neighbour elements move since they are connected like a chain. If the moved element stays within the boundaries of the neighbours, the neighbours do not have to be moved. If the moved element violates the boundaries of its neighbours, the neighbours have to be moved to satisfy the boundary constraints again. If a neighbour is move, then its neighbours are moved too if necessary. Like this, the movement of one element is locally propagated through the object.

The ChainMail algorithm consists of two steps to calculate the deformation that occurs when an element is moved. The first step calculates the movement of the neighbour elements of the moved element. If any neighbour element has been moved, the movement of their neighbours is calculated too and so forth. In the second step, a relaxation step, the object is relaxed by locally adjusting the elements until the object reaches a valid state of minimum energy.

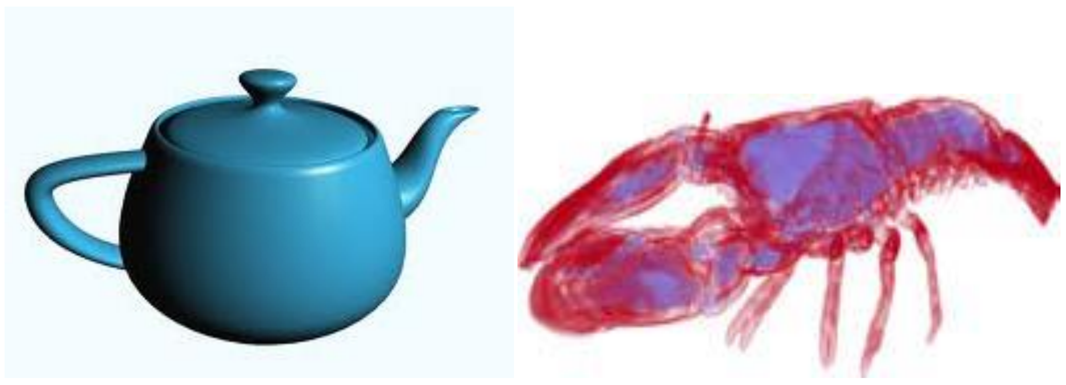
ChainMail has the advantage that its complexity does not grow with the number of elements of the object but only with the number of affected elements. The performance of the algorithm is based on two features of the algorithm: 1, the deformation is calculated depending on simple constraints; 2, each element of the dataset is processed at most once per deformation step.

A basic discussion of volumetric data will be given in the next section, and then followed by a detailed description of our implementation of ChainMail algorithm in the following section.

3.3.1 Volumetric Data

3.3.1.1 Basic Data Structure

Interactions with most objects in the real world are through their surfaces, for example, the skin of our body and the green surface of the leaves on the trees. A volume describes not only the surfaces of a geometrical object, but also the full distribution of information inside these surfaces. Examples are the air pressure or air flow around the body of a car, or the value of some tissue parameter throughout the body of a human patient. The two examples in Figure 3.14 illustrate the difference between non-volume and volume visualizations.



(a) (b)

Figure 3.14: Two example of visualization. (a) is an example of visualization of non-volume (http://www.wikipidia.org) and (b) is an example of volume visualization (http://www.volvis.org). Observe how translucency in the volume visualization reveals internal features not present in the visualization of surface only.

A volume is essentially a function in 3D space. A scalar volume is a mapping from a position in space to a scalar parameter (V: R^3->R) and a vector volume maps the position to a vector property ($\vec{V}$: R^3->R^N where N is typically > 3). Volumetric data can, at any observed scale in the real world, be considered continuous down to an unmeasurable granularity. In computers the storage space is limited, so generally volumes in computer science are not defined continuously throughout the occupied space, but only are collections of discrete sample points called voxels. A continuous volumetric dataset must be defined by one or set of analytical functions that can be numerically estimated at any position. Examples of volumetric data used in the work presented in this thesis are CT and MRI data which are sampled scalar volumes. Volumes that are sampled by these techniques can simply be interpolated into a continuous function using interpolation.

A two-dimensional raster image can be used as the role model for the discrete representation of a volume. An image consists of *pixels* (short for “picture elements”) that are organized in a regular array. Pixels are the data elements of a 2D image, holding colour values. A discrete volume dataset can be represented in a similar fashion by just “lifting” the description from two dimensional to three dimensions. A 2D pixel is extended to a 3D *voxel* (short for “volume element”). Voxels are points in 3D space, along with an interpolation scheme that fills the in-between space, organized in a regular 3D array, covering the volume dataset. The definition of a voxel is compatible with a grid-based description of the volume dataset: voxels serve as grid points of a *uniform* grid. The grid points are connected by edges, forming hexahedral (i.e., cube-like) cells. In fact, a 3D uniform grid can be defined as a collection of grid points that are connected to form rectangular, hexahedral cells of equal size. Figure 3.15 illustrates a volume dataset represented on a discrete uniform grid.

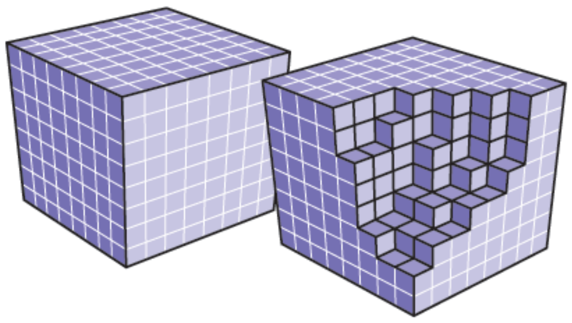


Figure 3.15: Volume dataset given on a discrete uniform grid.

A uniform *n*-dimensional grid has the advantage of being well-structured, which leads to a compact representation in computer memory (e.g., in the form of an *n*-dimensional array) and a fast access to selected data cells. Uniform grids, however, are not very flexible. Therefore, other grid structures may be used to represent discretized data. Higher flexibility can be achieved by permitting more flexible cell types or a more flexible combination of cells to a grid. For example, distorted hexahedral (i.e., with non-perpendicular cell axes) or completely different cell types (e.g., prisms) may be used. Moreover, different kind of cells may be combined in a single grid. Since volume data representations from CT and MRI are arranged in a rectilinear grid naturally, we will only focus on uniform grid for simplicity and efficiency in this thesis.

3.3.1.2 Visualization of Volumetric Data

Visualization is the process of making something perceptible to the mind or imagination. In computer science this refers to the rendering of data onto a display to convey a message or make understandable some abstract features contained in the data. This display is often a monitor showing visual images, but may also be a haptic display providing a haptic representation of the data, loudspeakers conveying auditory cues or, in the future, even an olfactory display. Volume visualization is the art and science of presenting the information that resides in volumetric data, or volumes.

The amount of information contained in a volume can be immense and it is often advantageous to process the data to make important features more easily perceivable. The process of visualizing volumetric data includes tasks such as *data reduction* to lower the amount of displayed data, *data extraction* to automatically select parts of the data that may be of interest, *data enhancement* to improve the definition of unclear but potentially important features, *data representation* to convert data into a form that can be shown on the designed display. The main aim is to convert the data from an abstract cloud into a representation that can be perceived and understood, and thereby make best use of the human analytical system.

The full data of a volume should not be rendered with full opacity. That would produce the visual impression of a solid cube where only the data at the six sides are visible. Much of the process of creating a visual representation of volumetric data is to remove unwanted and unnecessary part in the data and enhance the important features through a more sparse visual representation. In this way unimportant, redundant and occluding information in the volume is removed to better show the important parts.

A typical example of removing data is the extraction and rendering of iso-surfaces in scalar volumes. Many algorithms exist to extract from scalar data an explicit iso-surface geometry, a geometry that can then be rendered on the screen. The rendering of geometrical representations of volumetric data is a type of indirect volume rendering. As such it suffers from the disadvantage that it only represents a pre-selected subset of the data, in this case at positions defined by a simple isovalue. A more powerful approach to produce a visual representation of volumetric data is the *direct volume rendering* (DVR). In this approach the full data is considered, but only parts that are important are rendered by manually or semi-automatically making uninteresting parts transparent or semi-transparent. In this way the interesting parts stand out and can be visually identified or explored by a user of the system. The DVR approach can, by considering the full volume, potentially show properties that cannot be represented by simple iso-surfaces such as the thickness of a surface. This can represent, for example, the thickness of the skin in medical visualization.

It is common in volume rendering that a set of *transfer functions* is used, each mapping one scalar value to another (τ: R->R), to map the scalar value of the volume to the red, green, blue and opacity (alpha) components of the visual rendering. The transfer functions are thus used to specify the visual impression of different values in the volume, and which values should be transparent and how transparent they should be.

Volume rendering techniques can be classified as either image-order or object-order methods. Image-order approaches iterate over the pixels in the image to be produced, rather than the elements in the scene to be rendered. The data volume is traversed beginning at pixels. On the other hand, object-order methods follow some organized scheme to scan the 3D volume in its object space. The traversed volume areas are then projected onto the image plane. Some classical volume rendering approaches are briefly introduced below.

Ray casting. Ray casting is the most popular image-order method for volume rendering. The basic idea is to directly evaluate the volume-rendering integral along rays that are traversed from the camera. For each pixel in the image, a single ray is cast into the volume (neglecting possible super-sampling on the image plane). The volume data is resampled at discrete positions along the ray. Figure 3.16 illustrates ray casting.

The natural traversal order is front-to-back because rays are conceptually started at the camera. Ray casting is the most important method for CPU volume rendering; it has been used for quite some time, and many acceleration methods have been developed. With today's powerful graphics cards, GPU ray casting has experienced great development as ray casting is very easy to parallelize. GPU ray casting has the advantages that it can be easily extended to benefit from acceleration techniques and that it supports both uniform grids and tetrahedral grids. Therefore, GPU ray casting has already become very popular within a short period of time – and it is safe to assume that ray casting will play an even more important role as GPUs further evolve.

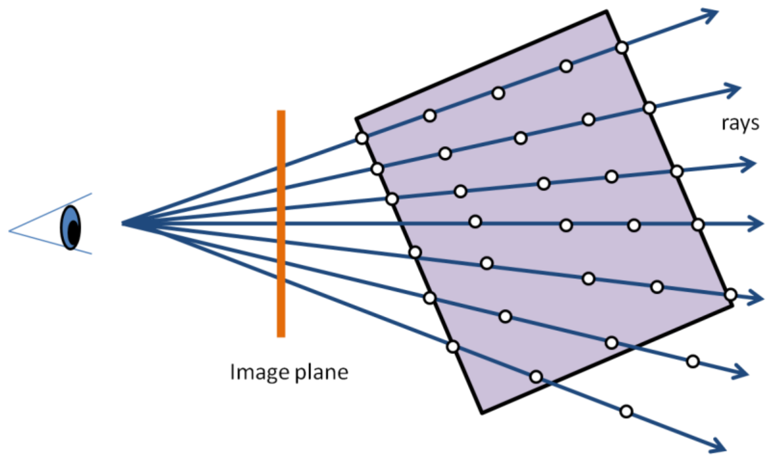


Figure 3.16: Ray-casting principle. For each pixel, one viewing ray is traced. The ray is sampled at discrete positions to evaluate the volume-rendering integral.

Texture slicing. Texture slicing is an object-order approach: 2D slices located in 3D object space are used to sample the volume. The slices are projected onto the image plane and combined according to a compositing scheme. Slices can be ordered either in a front-to-back or back-to-front fashion – and the compositing equation has to be chosen accordingly. Texture slicing is directly supported by graphics hardware because it just needs texture support and blending (for the compositing schemes). Therefore, texture slicing is widely available and very efficient. One drawback, however, is that it can only be used with uniform grids.

Shear-warp volume rendering. Shear-warp volume rendering is strongly related to 2D texture-based slicing. In this object-order method, the volume is traversed in a slice-by-slice fashion. The basic idea of shear-warp is illustrated in Figure 3.17 for the case of an orthogonal projection. The projection does not take place directly on the final image plane but on an intermediate image plane, called the base plane, which is aligned with the volume. The volume itself is sheared in order to turn the oblique projection direction into a direction that is perpendicular to the base plane, which allows for a fast implementation of this projection. In such a set-up, an entire slice can be projected by 2D image resampling. Finally, the base plane image has to be warped to the final image plane. Note that this warp is only necessary once per generated image, not once per slice. Perspective projection can be accommodated by an additional scaling of the volume slices.

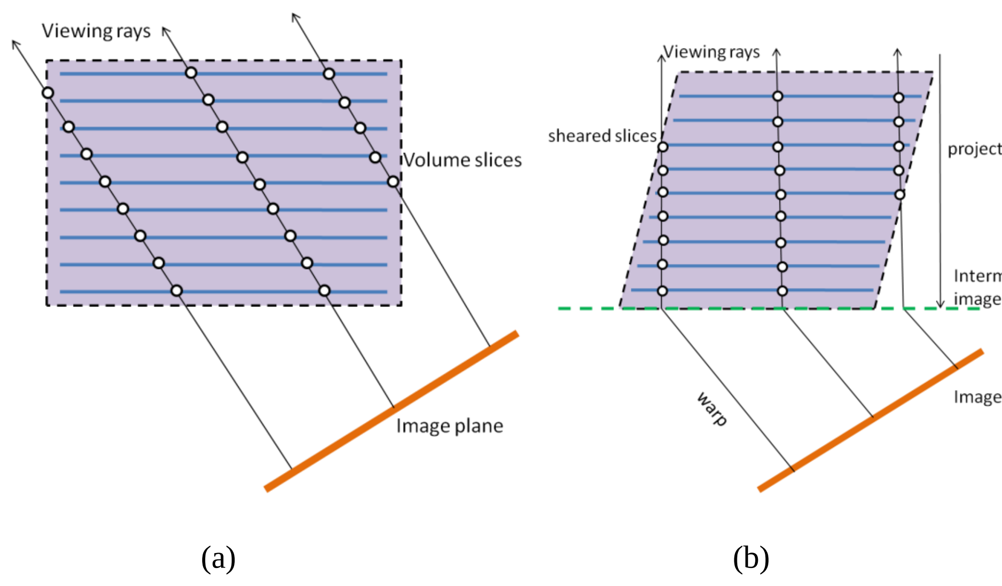


Figure 3.17: (a) The standard transformation. (b) shear-warp factorized transformation for a parallel projection.

Two-dimensional texture slicing is directly related to shear-warp volume rendering. When 2D textures are used to store slices of the volume data, and a stack of such slices is texture-mapped and blended in graphics hardware, bilinear interpolation is also substituted for triliner interpolation, similar to shear-warp. The difference between shear-warp rendering and 2D texture slicing is the order of performing the image warp and the compositing: texture slicing warps each slice and performs compositing on the final image, whereas shear-warp rendering only warps the intermediate image once.

A strength of the shear-warp algorithm is the possibility for optimizations. The shear-warp algorithm with run-length-encoded volume is one of the fastest CPU-based speed acceleration techniques developed so far for direct volume rendering (Choi *et al.*, 2012). However it has some defects too, such as the increases in memory consumption and preprocessing time as well as the deterioration in image quality. By making direct access to the memory space where initially loaded volume data is stored, image quality could be enhanced and memory consumption could be decreased without reducing rendering speed. By creating only one run-length-encoded volume and by combining non-photorealistic rendering techniques with shear-warp algorithm, those defects could also be solved. All these implementation could also be applied on not only a GPU using API such as CUDA but also various parallel techniques (Sathappan *et al.*, 2011; Surendran *et al.*, 2011) to get further speed acceleration.

Splatting. The idea of splatting (Zhang *et al.*, 2011) is to project 3D reconstruction kernels onto the image plane. The 2D image of such a 3D kernel is called a footprint. Splatting is an object-order approach: it traverses the volume in object space and projects volume elements onto the image space. In general, splatting allows for a quite flexible spatial order for traversing the volume. For example, it might be applied to traverse a uniform grid in a voxel-by-voxel fashion, or it might even be applied to scattered data (i.e., a cloud of arbitrarily distributed data points) – as long as some spatial sorting is provided to guarantee a correct result for back-to-front or front-to-back compositing. Image-aligned sheet-based splatting chooses a specific order of traversal by sampling the volume along sheets (i.e., slicing slabs) that have the same orientation as the image plane.

Cell projection. Cell projection (Marroquim *et al.*, 2008) is an object-order approach for the volume rendering of tetrahedral grids or even more complex unstructured meshes. The first cell projection algorithm that made efficient use of graphics hardware is the projected tetrahedral (PT) algorithm by Shirley and Tuchman (1990).

The basic idea of the PT algorithm is to traverse the cells of the unstructured grid and project these cells onto the image plane. The projection itself leads to a collection of triangles that represent the image of the 3D cell on the image plane. The PT algorithm consists of the following steps.

1. Decomposition of the unstructured grid into tetrahedral cells.
2. Spatial sorting of the cells according to their distance from the camera.
3. Classification of each tetrahedron according to its projected profile, along with a decomposition of the projected tetrahedron into triangles (on the image plane).
4. Assignment of colour and opacity values attached to the triangles.
5. Rendering and blending of the triangles.

Unfortunately, cell projection with the emission-absorption model of volume rendering is connected to noncommutative blending (compositing). Therefore, it requires a view-dependent depth sorting of cells, which still has to be performed on the CPU. Whenever the camera or the volume is moved, new graphical primitives have to be generated by the CPU and transferred to the GPU. Therefore, cell projection benefits only in part from the performance increase of GPUs. Recent work by Maximo *et.al.* (2010) is exclusively based on the rasterization of simple geometric primitives and takes full advantage of graphics hardware. Both vertex and geometry shaders are used to compute the tetrahedral projection, while the volume ray integral is evaluated in a fragment shader; hence, volume rendering is performed entirely on the GPU within a single pass through the pipeline.

We choose 3D texture mapping volume rendering as our volume visualization technique since it is very efficient and suits our situation where only uniform grids are considered. Texture-based volume rendering technique performs the sampling and compositing steps by rendering a set of 2D geometric primitives inside the volume. Each primitive is assigned some texture coordinates for sampling the volume texture. The proxy geometry is rasterized and blended into the frame buffer in either the back-to-front or front-to-back order. In the fragment shading stage, the interpolated texture coordinates are used for a data texture lookup. Next, the interpolated data values act as texture coordinates for a dependent texture lookup into the transfer function textures. Illumination techniques may modify the resulting colour before it is sent to the compositing stage of the pipeline. Figure 3.18 show two examples of our rendering of some volumetric data using direct volume rendering: (a) and (c) show the whole volume, while (b) and (d) show only the bones by using a specific transfer function.

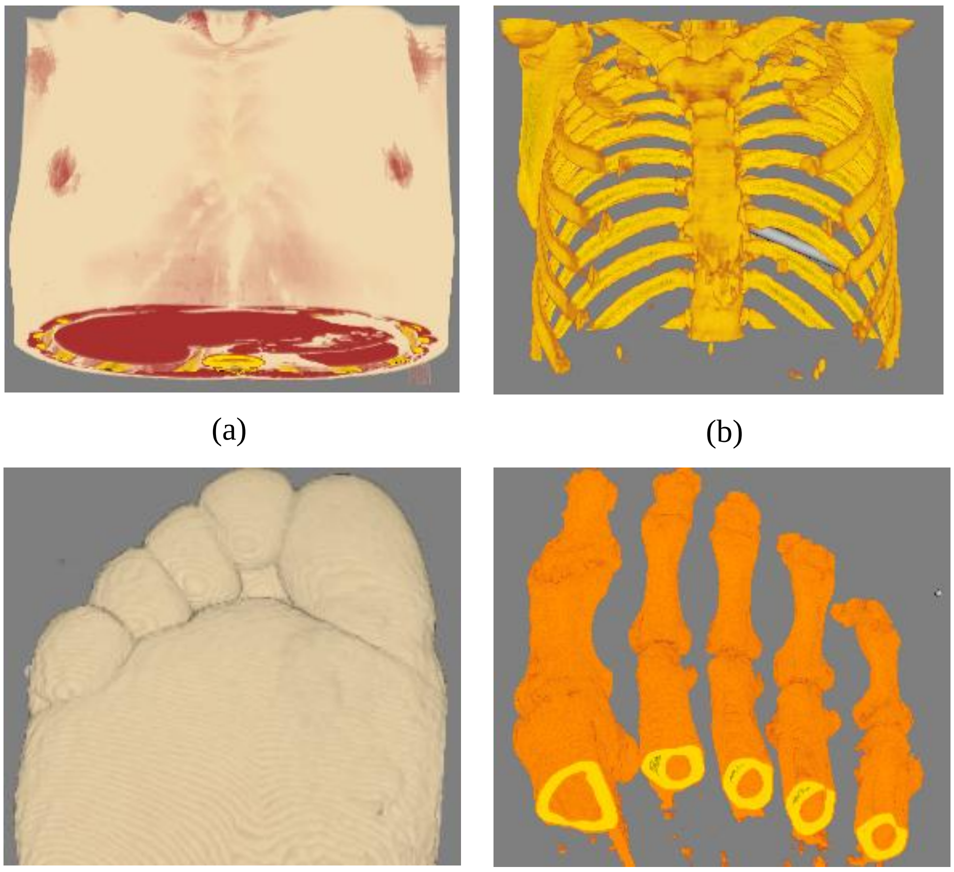


Figure 3.18: Direct volume rendering.

3.3.2 ChainMail Algorithm for Groove Generation

This section provides a more accurate explanation of the ChainMail algorithm and its implementation. The presented implementation is based on the *original 3D ChainMail* algorithm and *Enhanced ChainMail* algorithm, since this deformation system is designed to handle inhomogeneous material. The *Generalized ChainMail* algorithm (Li and Brodlie, 2003) is not used for our approach since in this project the modelled voxels lie in a rectilinear grid. The extension to a non-rectilinear grid as proposed by the Generalized ChainMail algorithm is not necessary here.

3.3.2.1 The ChainMail Neighbour Constraints

ChainMail deals with volumetric data initially ordered in an axis-aligned three-dimensional grid. The grid structure induces axis-aligned neighbourhood relationships which are spatially defined by the ChainMail constraints. Each chain element defines the so-called *valid areas* for its neighbours as depicted in Figure 3.19. As long as the neighbours remain inside their respective valid areas they remain static. If an element is outside its valid area, it is considered to have violated its constraint and has to be adjusted.

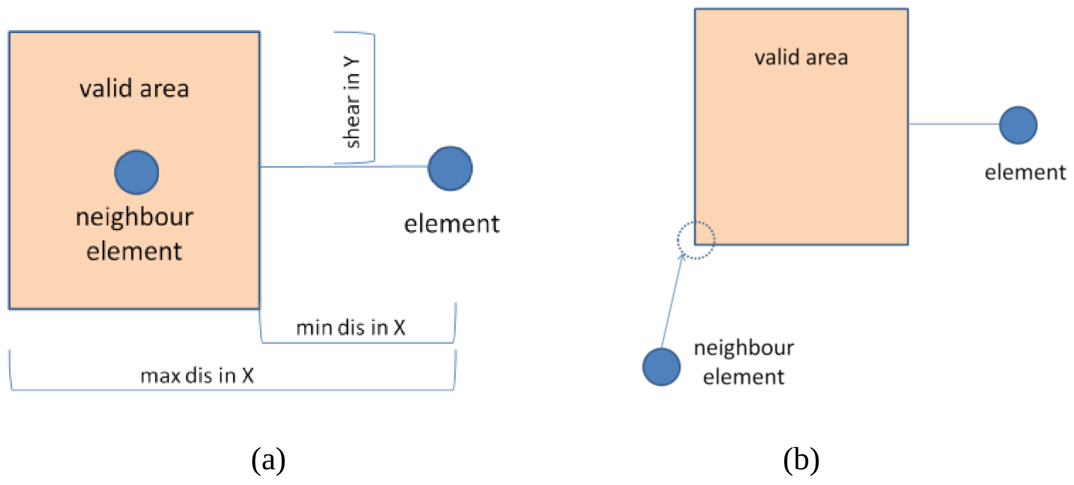


Figure 3.19: (a) ChainMail constraints. (b) Constraints violation

Each element defined valid areas for each of its six neighbour elements. The valid areas are described by the following attributes:

- minimal distance
- maximal distance
- allowed shear

All attributes are defined for each principal axis separately, so altogether nine attributes are needed. This axis dependent property leads to the ability to define anisotropic materials. Each valid area is defined on a fixed side of the element. Rotation of the element and its valid area is not possible. This limits the deformation algorithm to rather simple deformations since any rotational behaviour like bending a structure is disabled. However this would not be a problem is our project as we only need the deformation algorithm for generating groove which is normally in a simple shape and no rotational behaviour are involved.

3.3.2.2 Element Processing

The deformation cycle of the ChainMail algorithm starts with an initial movement. Commonly it is caused by a user interaction such as dragging of an element with the mouse, or pushing with a haptic device. In our framework this initial movement is invoked by haptic interaction. After the first element is moved all of its six neighbours have to be checked to see whether they remain in their valid areas defined by the moved element. If not they have to be adjusted accordingly.

The Enhanced ChainMail algorithm stores all elements which need to be adjusted (from now on called *candidates*) in a priority queue. The key value for the queue is normally the amount of constraint violation. Elements with a large violation will be processed first. In our implementation the violation value is defined as the minimal square distance between the elements of the corresponding valid area. For two horizontal neighbour elements with the positions v_1 and v_2, the constraint violation is calculated as follows:

$$\textit{amount of constraint violation} = \textit{distance}(v_1,v_2) - \textit{max}_{dx} \qquad (3.9)$$

where max_{dx} stands for the maximum distance in X (*max dis in X*) in Figure 3.19.

Note that each element belongs to six valid areas defined by its six neighbours. Any violation of the six valid areas causes the element to be adjusted because every neighbour defines the constraint for the element. The actual valid area is the area defined by the element (we call it *sponsor*) that caused the insertion to the priority queue of the actual element.

Figure 3.20 shows how the algorithm performs on the elements in the 1D case. In the first step, element *J* is moved to the right most position (red dotted circle) and forces the left neighbours to be dragged in order to satisfy the region constraints. Note that the drag necessary (red dotted arrow) in the third step is smaller than that in the second step. In the fourth step no drag is necessary and the deformation terminates.

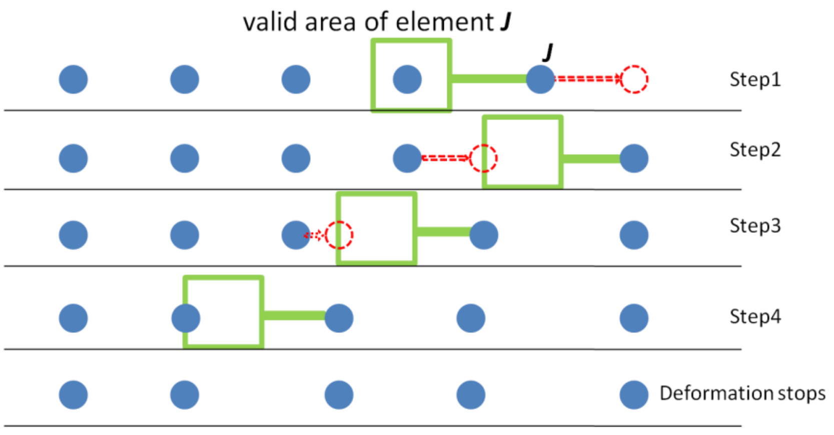


Figure 3.20: A sample deformation of a 1D chain.

3.3.2.3 Multiple Element Processing Avoidance

One main reason for the efficiency of the ChainMail algorithm is that each element is processed at most one time per deformation step. However, the algorithm trades correctness for speed and does not assure validity under every circumstance. In the 2D case as shown in Figure 3.21, the upper left element violates the constraints of two neighbours in step 4. In this situation the upper left element could be added to the candidate queue twice, one by its right and one by its bottom neighbour. In order to maintain the efficiency, a method is needed to avoid multiple processing of elements.

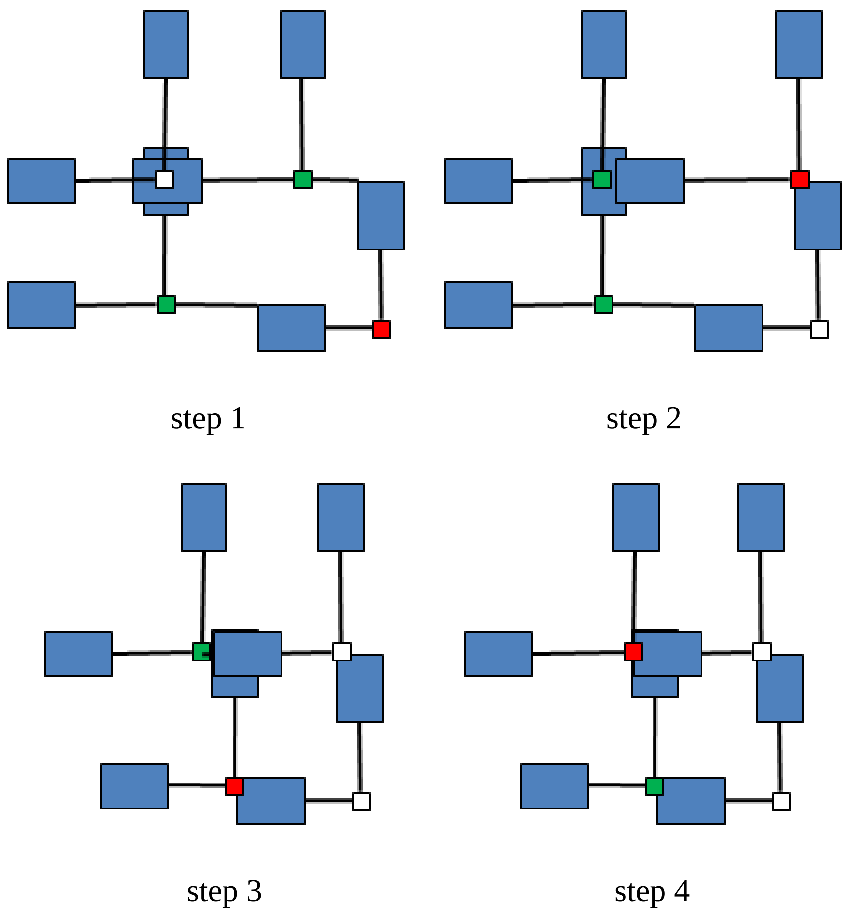


Figure 3.21: A sample deformation of a 2D chain. Red marked elements are actually moved while the green elements are members of the candidate queue. Valid areas (blue) are only drawn in the directions necessary.

In the original ChainMail algorithm, this problem is solved by managing six different candidate lists, one for each principal direction (*left, right, top, bottom, back, and front*). The lists are processed in turns and each list contains a different group of neighbours which assures that each element can only be added to one list. In the case of a 2D ChainMail implementation, the list for the right direction would only be able to add right, top and bottom neighbours. Similarly the list for the left direction only accepts the left, top and bottom neighbours. For the top direction it would be allowed to add only the top neighbours and similar for the bottom, back, and front directions.

The Enhanced ChainMail algorithm preserves the feature that each element needs to be processed just once. However the six lists are replaced by one priority queue. Another method is needed to ensure that each element is processed just once.

Unfortunately it is not explained how this problem is solved in the original paper for the Enhanced ChainMail implementation (Schill *et al.*, 1998).

In the work by Dräger (2005), this problem is handled by an *ignore direction flag* which adds the elements into the candidate queue in a similar way as in the original ChainMail algorithm. For each element a set of flags defines which neighbours will be ignored even though they should be added to the candidate queue. These flags are individually set if an element is added to the queue depending on the direction of the *sponsor*. The rule which defines the setting of the flags has to ensure that elements are not added multiple times. Figure 3.22 shows an example of this ignore direction flag mechanism for the 2D ChainMail grid. The rules are:

- If the candidate is sponsored by the left neighbour: ignore its left neighbour
- If the candidate is sponsored by the right neighbour: ignore it right neighbour
- If the candidate is sponsored by the top neighbour: ignore its left, right, top neighbours
- If the candidate is sponsored by the bottom neighbour: ignore its left, right, bottom neighbours

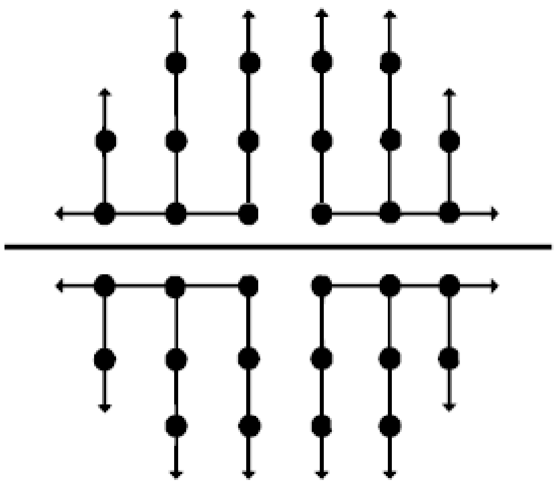


Figure 3.22: The ignore direction flag mechanism from Dräger (2005).

Unfortunately this method contradicts with the original intension of the Enhanced ChainMail algorithm. The Enhanced ChainMail algorithm should enable a shock wave like deformation propagation through inhomogeneous material by a constraint violation processing order. The ignore direction flag mechanism restricts the adding of neighbours dramatically.

In the work by Schulze (2006), the problem is solved through a simple timestamp mechanism. Each time the ChainMail algorithm is invoked the timestamp is incremented. Each element has a local timestamp. If an element should be added to the queue, the local timestamp is first compared with the global timestamp. The element will be added if these two numbers are not equal. Afterwards the local timestamp is set to be the global so it cannot be added a second time. But the method suffers from the fact that the timestamp will overflow after awhile.

Instead of realistic shock wave like deformation, we only need ChainMail for fast and simple deformation for the groove generation. The ignore direction flag mechanism fits our purpose very well.

3.3.2.4 Move Multiple Elements

The original ChainMail and the Enhance ChainMail algorithms addressed only the manipulation of a single element. Often it is not appropriate to model interactions by manipulating only one element in most deformation systems. One single element represents just a very small part of the volume. On the other hand the simulated interaction tools are often in much bigger sizes than the size of a voxel. It is very common to move several elements at once.

This problem was also addressed in Dräger (2005) and solved by a so-called *multi move* mechanism. Every time a deformation was performed, the algorithm initially moved eight elements enclosing the centre of deformation. The ignore flags mechanism was adjusted to allow the movement of these eight elements at once. The algorithm is not able to do anything else other than moving these eight elements.

To overcome this inflexibility, the implementation described in Schulze (2006) was able to handle an arbitrary amount of initially moved elements. The ChainMail solver is initialized with a list of initially moved elements. Each of these elements is scheduled for deformation by pushing it into the candidate queue with the largest possible violation value. The high violation value assures that the element is processed first. The Enhanced ChainMail algorithm continues to process the candidate queue as described in section 4.3.2.

However, allowing the movement of several elements at once leads to the possibility of unpredictable behaviours. This can happen for example if neighbouring elements are moved into different directions. Such situations would not be prevented by the deformation system implemented in Schulze (2006). In their system, the users have to be responsible to forward consistent movements. For our application, the deformation behaviour is very simple and predictable, i.e., pulling the two sides of the cutting path along the perpendicular directions to generate the groove, so we can just restrict the application to single element moving. This initial moved element is just the nearest voxel to the contact position between the volume and the haptic probe in our implementation.

3.3.2.5 ChainMail Deformation Results

The Enhanced ChainMail algorithm enables efficient deformation of volumetric datasets. An example of this deformation method is depicted in Figure 3.23. A performance measurement is given in Figure 3.24. The plot confirms the complexity estimation of the ChainMail algorithm of *O(n)* since the computation time increases linearly with the number of moved elements. Depending on this measurement an average performance of 7.3156 * 10^5 *elements/seconds* can be estimated. The timing tests were performed on an Intel Xeon 2.53 GHz machine within 2GB RAM.

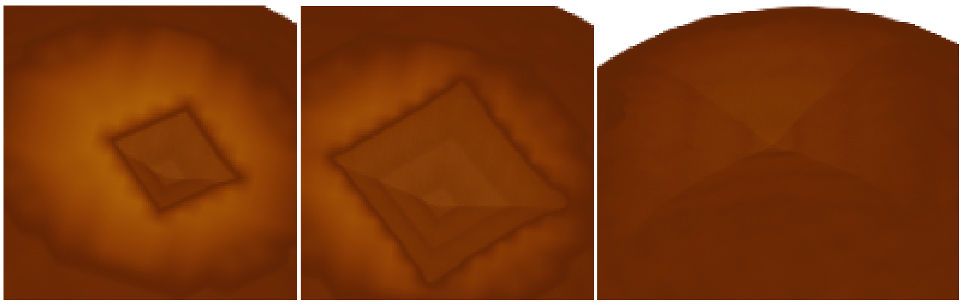


Figure 3.23: Results of ChainMail deformation.

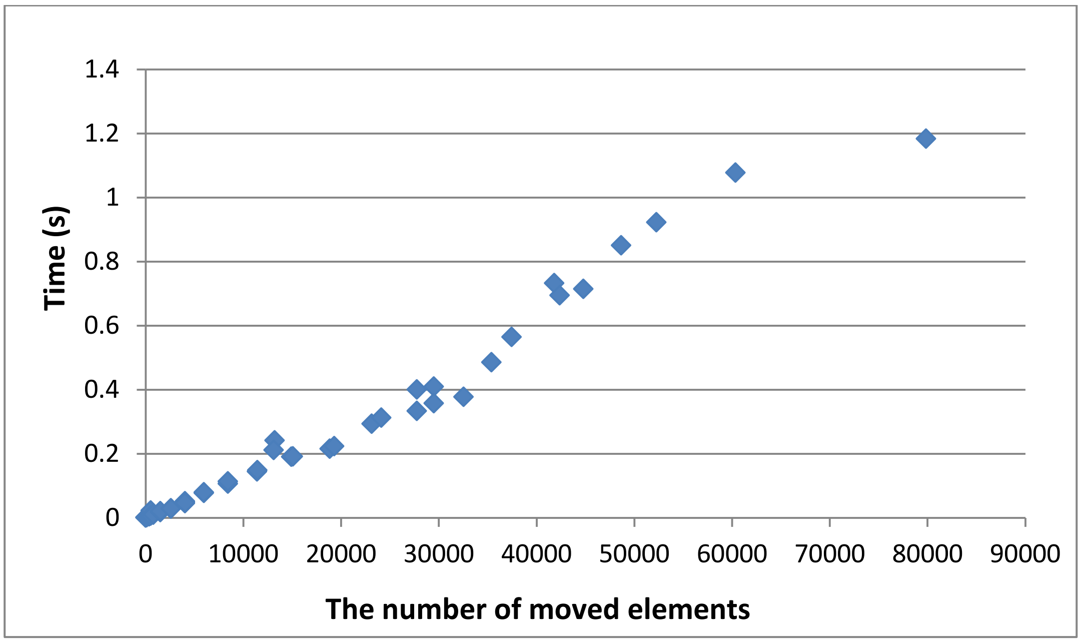


Figure 3.24: Scatterplot depicting timing test of the described ChainMail implementation. The x axis denotes the amount of moved elements, and the y axis denotes the computation time in seconds.

The deformation results in Figure 3.23 above show a very obvious artefact of the axis-aligned edges on the surface. This is caused by the uneven deformation propagation of the ChainMail algorithm. In the examples a homogeneous material is used, so it could be estimated that deformation is propagated circularly around the centre of deformation. The Enhanced ChainMail algorithm only assures an even deformation - propagation along the main axis. It depends on the strict axis-aligned neighbourhood constraints of the ChainMail data structure and results in rhombus like deformation propagation.

This artefact is results from the ChainMail algorithm itself since it focuses on fast deformation of volumetric data. Relaxation can be performed afterwards to compute physically based volume deformation in most deformation systems. Relaxation is a method which tries to solve a global optimization problem by iteratively solving local optimizations. The goal is to minimize the global error which is defined as

$$E = \sum_i e_i \tag{3.10}$$

The global error E is the sum of the local errors e_i of element i. Figure 3.25 shows the results in Schulze (2006) before and after the relaxation. It can be clearly seen here that the artefact is largely overcome by the relaxation.

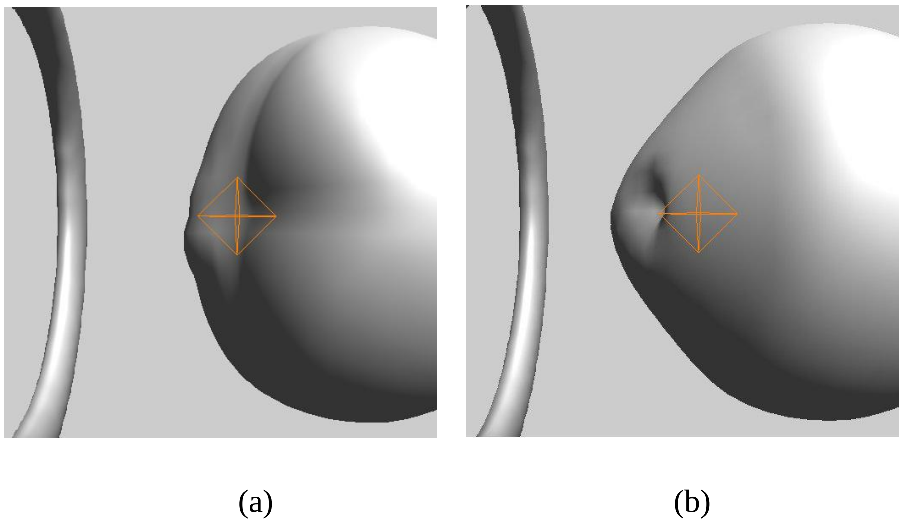


Figure 3.25: (a) ChainMail solution. (b) Solution through relaxation with springs.

As a matter of fact, this artefact would not cause too much problem in our application. Realistic and smooth deformation and cutting have already been achieved on the outer surface model. The ChainMail algorithm is only needed for fast deformation of the underline volume for generating grooves, and to reveal the internal heterogeneous structures and materials. Thus the time-consuming relaxation would not be necessary for our case. By unlinking the neighbouring elements along the cutting path and using the ignore direction flag mechanism, the deformation is constrained as only propagated along the desired directions, i.e. the deformation on the left side of the cutting path only propagates along the left and bottom directions, while the right side deformation only propagates along the right and bottom directions. The volume is hence smoothly split along the cutting path to generate the groove, as shown in Figure 3.26. The arrows in the figure show the directions of deformation propagation. A memory management system is implemented for the local direct volume deformation as it is generally assumed that the deformation around the cutting site will remain local. Only the necessary parts of the volume are loaded into the memory for deformation calculation while the full information of the original data is still available for visualization. Figure 3.27 is a closed-up illustration of the process of pulling apart the tissues along the cut by our volume deformation algorithm. Figure 3.28 presents the results of groove generation on the torso dataset with different widths.

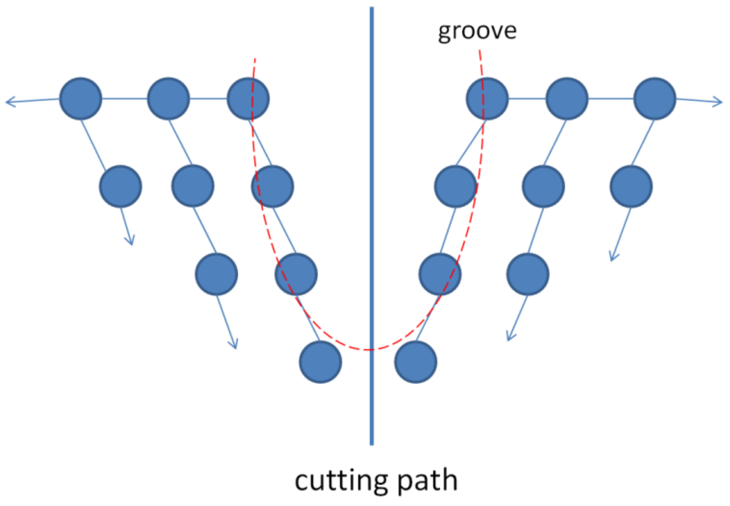


Figure 3.26: Groove generation.

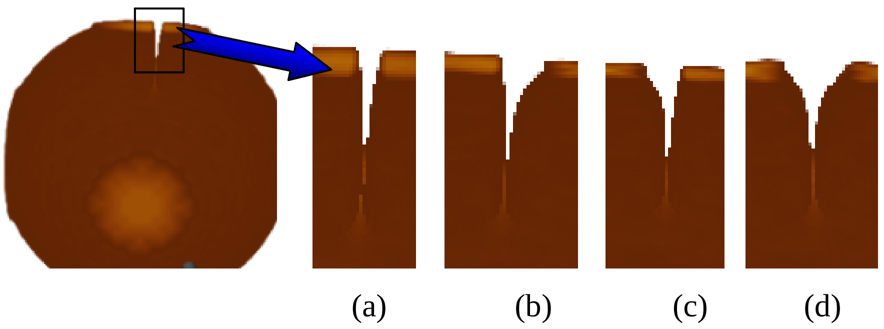


Figure 3.27: Groove generated from our ChainMail implementation without relaxation solution (a) A thin cut is made through a volume and the left and right neighbours along the cut are unlinked. (b) Right side is pulled away. (c) Left part is pulled away. (d) Left and right sides are being pulled away simultaneously and formed the cut groove.

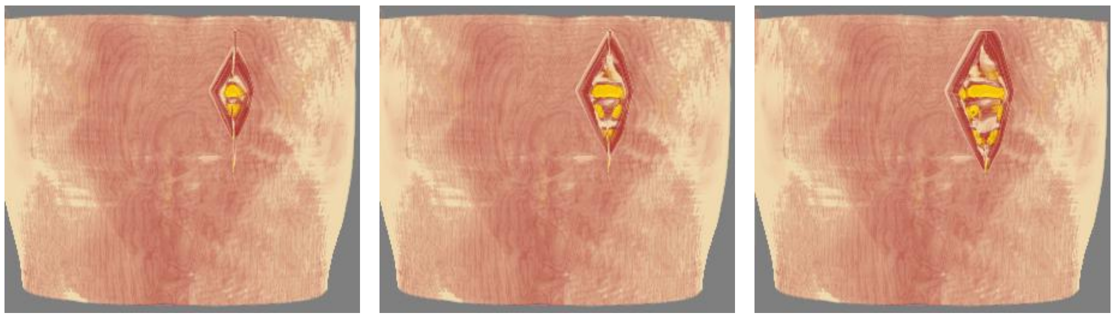
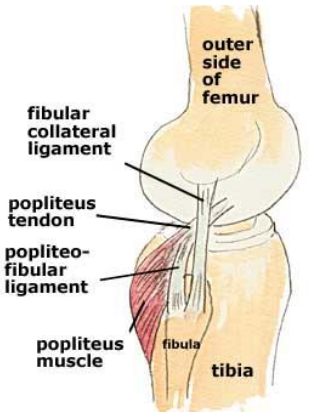
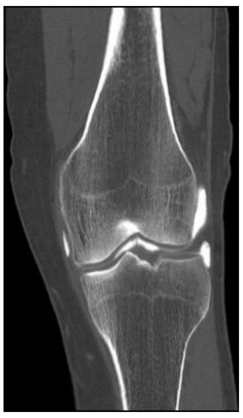


Figure 3.28: Groove generated on the torso model.

3.3.3 Drill Effect

Our volumetric deformation model can be extended easily to simulate other operations such as drillings in a bone surgery. Bone drilling is a very common bone surgical procedure, which is often a preliminary step for the insertion of pins or screws during the repair of a bone fracture or installation of a prosthetic device. Precise bone preparation is a key element for the successful long-term fixation of orthopaedic implants. Initial stability leading to reduced micro-motion and direct apposition of the bone against the implant are fundamental for the proper load transfer and bone remodelling. The fit and fill of the implant are created by shaping and sizing a cavity within the bone to accommodate the implant, which is usually accomplished by standard machining operations such as broaching, milling and drilling.

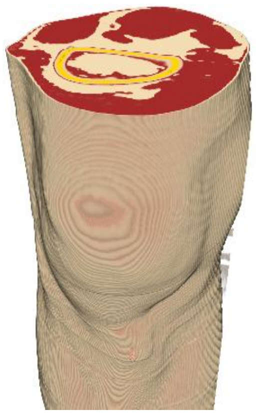


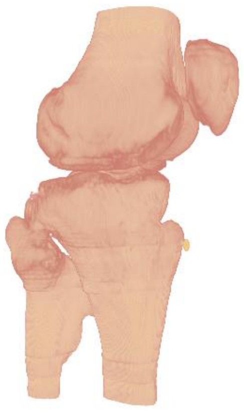


(a) (b)

Figure 3.29: (a) Anatomy illustration of a knee. (b) a CT image of knee.

We choose to simulate the bone drilling procedure in a knee surgery. A knee dataset from CT scan is used as shown in Figure 3.29. The volumetric data size is 186x229x305. The data is segmented into bone, muscle, fat and skin with the designed transfer function as illustrated in Figure 3.30.





(a) (b)

Figure 3.30: (a) Different materials of the knee are rendered by specific transfer function. (b) Only bones are displayed.

The drilling simulation is implemented by removing the voxels which are located inside the haptic stylus modelled as a sphere with a radius and a centre as parameters. If the distance of voxel and the stylus centre is smaller than the radius, the voxel lies inside the haptic stylus will be removed by setting its value to zero. If the distance is larger than the radius, no reaction will be applied to the voxel as it lies outside the stylus. Figure 3.31 illustrates one drilling result from our system.

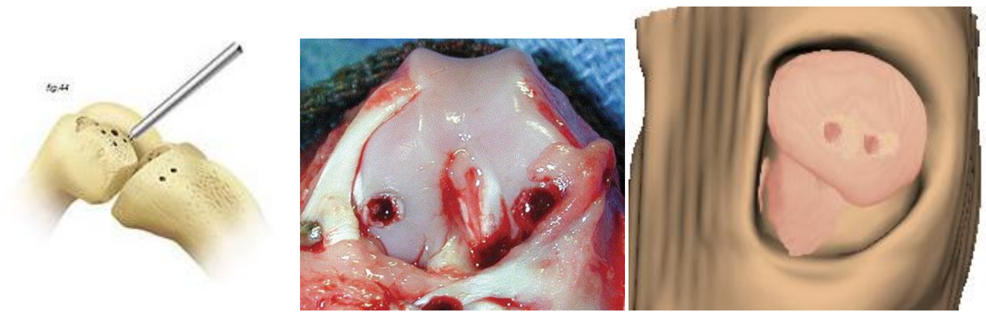


Figure 3.31: (a) Illustration of drill holes in a knee surgery, (b) drill holes on a limb knee and (c) drill holes created by our surgery simulation system.

Figure 3.32 shows some snapshots of drilling effect on another volume data with a size of 256*256*256.

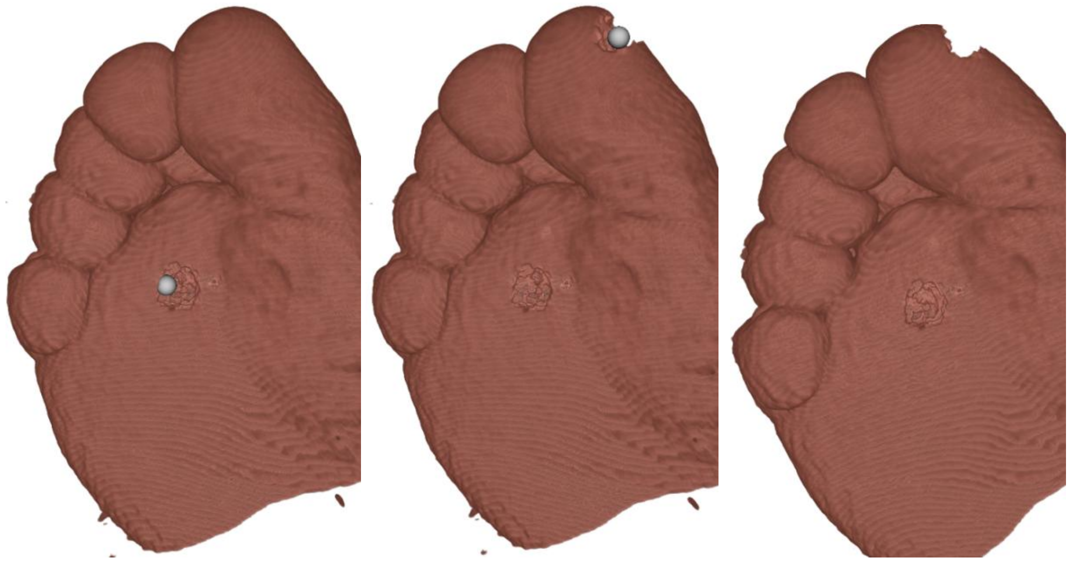


Figure 3.32: Drill effect on the foot dataset.

3.4 Hybrid Cutting

Although mesh-based methods are efficient for simulating simple cutting process on a surface, maintaining and adapting a mesh-based representation is less appealing in more complex scenarios where interior structures are important. Volumetric representations have enjoyed popularity due to their added flexibility in dealing with such situations. We propose a hybrid framework incorporating a volumetric model with its corresponding outer surface mesh. Our approach begins with a smooth progressive cutting on the outer surface mesh. It continues with the underline volumetric model for straightforward handling of groove generation and direct volume haptics interaction without the need for complex remeshing. So far we have achieved realistic surface cutting in Section 3.2 and groove generation on the volumetric model in Section 3.3. We will combine these two data models into a seamless hybrid model for progressive surgery cutting in this section.

3.4.1 System Overview

Out system is implemented in three stages, as shown in Figure 3.33. In stage 1, the haptic tool interacts with the outer surface model. Once there is a collision detected, the force feedback is calculated and compared to the yield limit of the material being simulated. If the force is within the limit, a physically based deformation (PAFF) model will be applied to the surface model, to calculate the deformation according to the haptic direction and haptic force feedback. If the haptic is push harder, the force will reach the limit and this will lead to stage 2 of the system.

In stage 2, the force feedback is beyond the yield limit and a rupture will occur. The cutting on the surface model is achieved by node-snapping algorithm and the cutting on the volumetric model is implemented by removing the voxels in contact with the haptic tool during haptic moving. Both processes are progressive and follow exactly the motion of the device. After this, a small crack on the surface model and a thin rupture on the volumetric model are formed for our hybrid model.

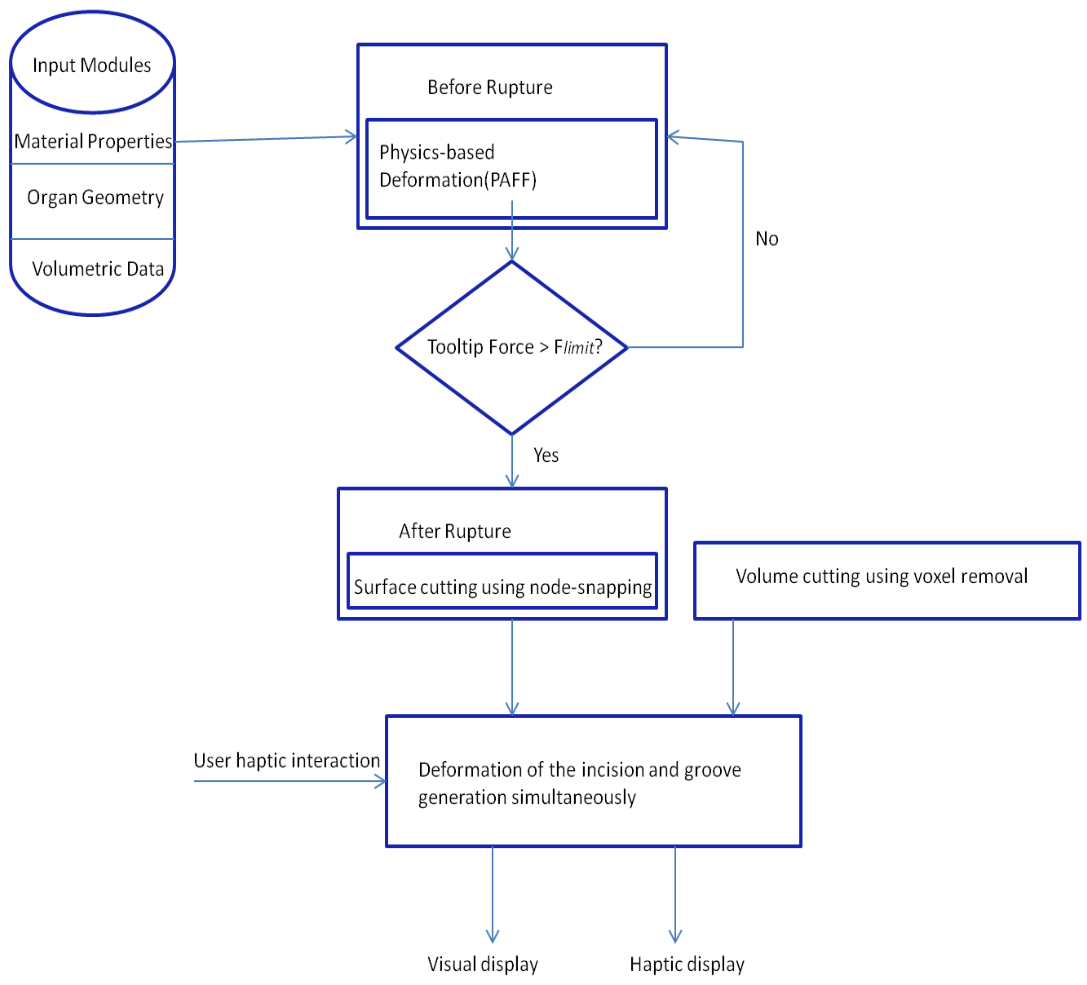

Input Modules
Material Properties
Organ Geometry
Volumetric Data
Before Rupture
Physics-based Deformation(PAFF)
No
Tooltip Force > F*limit*?
Yes
After Rupture
Surface cutting using node-snapping
Volume cutting using voxel removal
User haptic interaction
Deformation of the incision and groove generation simultaneously
Visual display
Haptic display

Figure 3.33: System Overview.

In stage 3, a pulling operation is applied to the hybrid model to reveal the region of interest. In a real surgery, after a thin cut on the object, the surgeon often uses a retractor to hold back soft tissues such as skin and muscle to open up the cut. The haptic device is modelled as a retractor in this stage to separate the cut. During the pulling, a wrinkle effect will occur on the surface model due to the elasticity of skin, and a groove will be generated by our modified ChainMail algorithm on the volume model to reveal the heterogeneous structures and material properties underneath the surface. Both processes are controlled by haptic interaction and synchronized with each other.

3.4.2 Iso-Surface Extraction by Marching Cubes Algorithm

To maintain the consistency between these two data models, the outer surface used in our system is an iso-surface extracted from the inner volumetric model using the Marching Cube algorithm (Newman and Yi, 2006).

The standard MC algorithm, as originally described by Lorensen and Cline (1987), takes as its input a regular scalar volumetric dataset. Such a dataset has a scalar value residing at each lattice point of a rectilinear lattice in the 3D space. The lattice point at row $y_i(V_i)$ and column $x_j(V_j)$ of slice S_k ($1 < k < n$, where n is the number of slices) is directly adjacent to the lattice points at row y_i and column x_j of slices S_{k-1} and S_{k+1}. The MC processes the volumetric dataset by considering the "cubes" C_l that make up the volume. The cubes are defined by the volume's lattice. Each lattice point is a corner vertex of a cube. Figure 3.34 illustrates a cube defined by lattice points in adjacent slices *k* and *k+1*. The lattice lines shown in Figure 3.34 (i.e., the line segments that join adjacent corner vertices) define the edges of the cube.

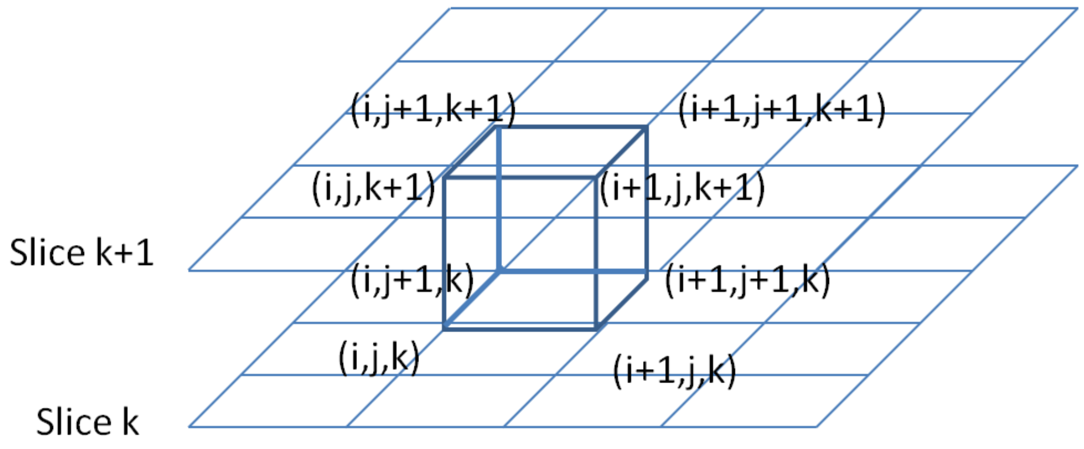


Figure 3.34: Illustration of a cube formed on lattice points.

The standard MC constructs a facetized isosurface by processing the dataset in a sequential, cube-by-cube manner in scanline order. The approach first processes the *m* cubes of the first row of the first layer of the dataset in sequential order: C_1, C_2, … , C_m. During this process, each cube vertex V_i that has a value equal to or above the isovalue $\boldsymbol{\alpha}$ is marked; all other vertices are left unmarked. The isosurface intersects each cube edge E_j between one marked vertex V_{js} and one unmarked vertex V_{je}. Any cube that contains an intersected edge is *active*. The computations for finding the active cubes can be viewed as the active cube determination component of the MC. This component can be implemented as a stand-alone early processing step or integrated with other processing, but in either case it involves dataset traversal in sequential, forward-marching order.

Since each of the eight vertices of a cube can be either marked or unmarked, there are 256 (2^8) possible marking scenarios for a cube. Each cube marking scenario encodes a cube-isosurface intersection pattern (i.e., configuration). However, the standard MC considers reflective and rotational symmetry, which results in just 15 marking scenarios. Cubes C and $\hat{C}$ are *reflectively symmetric* if each vertex at position V_i in C has the opposite marking as the vertex at the same position V_i in $\hat{C}$. Two reflectively symmetric cubes (and their cube-isosurface intersections) are shown at the centre and right (i.e., cubes A and A_F) in Figure 3.35. In the figure, circle symbols denote marked vertices. Cubes that are reflectively symmetric have the same cube–isosurface intersection pattern. Two cubes C and $\hat{C}$ are *rotationally symmetric* if there are some series of rotations R which, when applied to C, transform C to a new orientation in which the marking at each transformed vertex position V_i is identical to the marking at the same position V_i in $\hat{C}$. The cubes A and A_R in Figure 3.35 are rotationally symmetric. Rotationally symmetric cubes have equivalent cube-isosurface intersection patterns (the patterns vary only rotationally by the aligning transformation R).

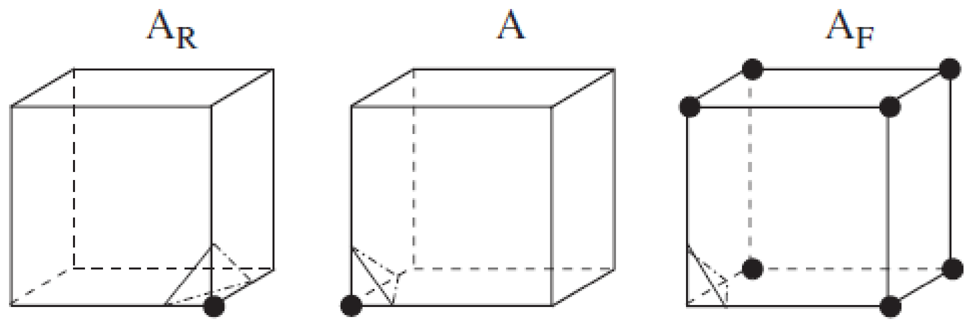


Figure 3.35: Illustration of reflective (A with A_F) and rotational (A with A_R) symmetries.

The 15 unique cube-isosurface intersection scenarios that result in when considering both of these symmetries are shown in Figure 3.36. We will use the topological case numbering of Figure 3.36, which is the same as the standard MC (Lorensen and Cline, 1987), throughout this thesis. For each scenario, the standard MC facetization of the intersecting isosurface is shown.

The isosurface-edge intersection locations can be estimated with subvertex accuracy using an interpolation technique. Standard MC employs linear interpolation to estimate the intersection point for each intersected edge. If a unit-length edge E has end points V_s and V_e whose scalar values are L_s and L_e, respectively, then given an isovalue $\boldsymbol{\alpha}$, the location of intersection $I = (I_x, I_{y,} I_z)$ has components of the form:

$$I_{\{x,y,z\}} = V_{s\{x,y,z\}} + \rho(V_{e\{x,y,z\}} - V_{s\{x,y,z\}}), \tag{3.11}$$

where

$$\rho = \frac{\alpha - L_s}{L_e - L_s} \tag{3.12}$$

One advantage of the cube-by-cube processing of standard MC is that each edge intersection location only needs to be computed once. Specifically, since each intersected internal edge E_j (i.e., E_j is not on the dataset boundary) is shared by four cubes, the point of isosurface-edge intersection I_j on E_j only needs to be computed for one cube C_a. The intersection point I_j can be reused during later processing of three other cubes C_b, C_c, C_d (where b, c, d > a) sharing edge E_j. Nevertheless, algorithms that use other traversal patterns can also achieve the same computational advantages via using supplemental data structures (e.g., hash tables).

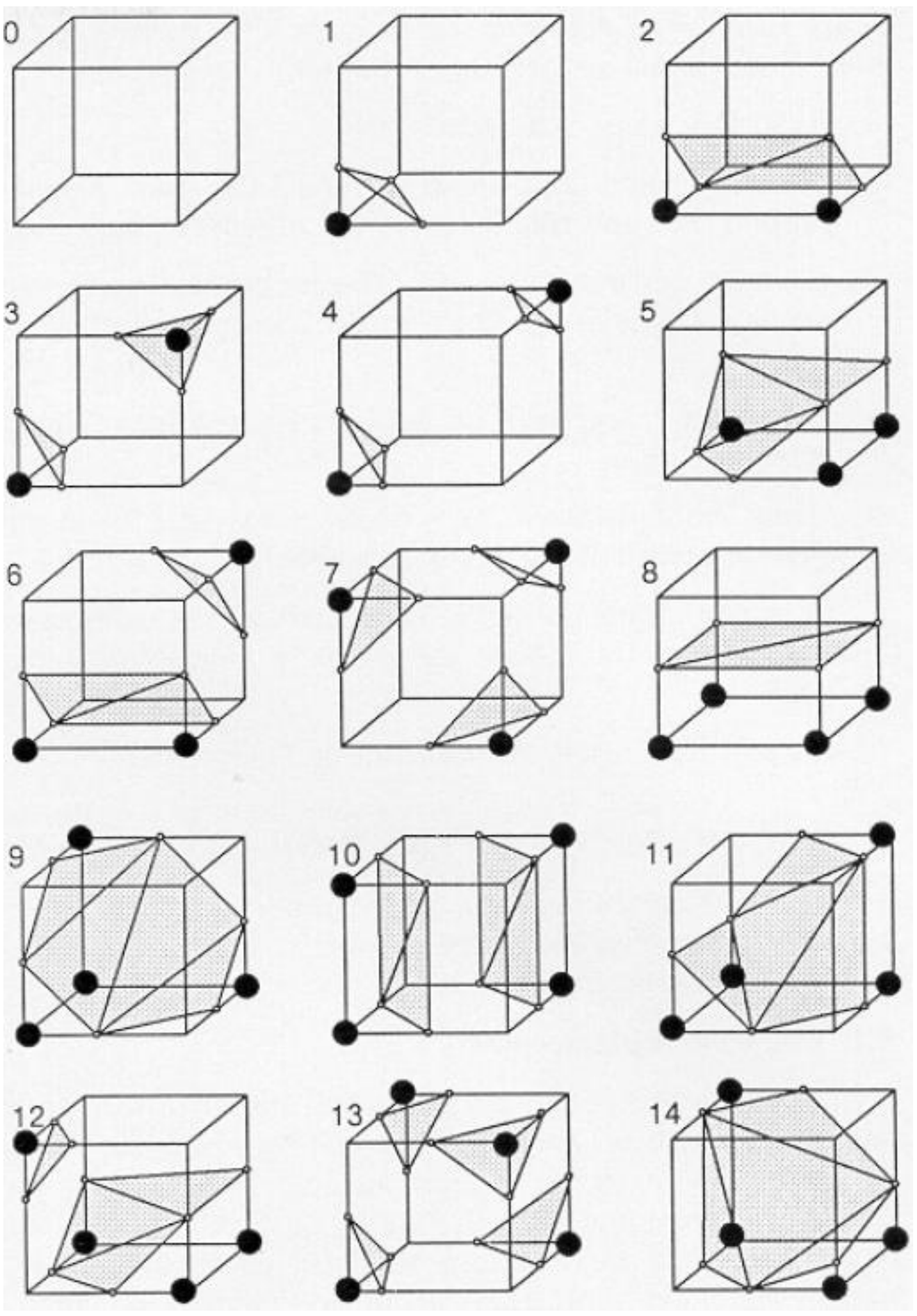


Figure 3.36: The 15 basic intersection topologies (using the numbering of (Lorensen and Cline, 1987))

The last step in MC is to generate triangular facets that represent the portion of the isosurface that intersects each cube. The intersection points define the vertices of the triangles, and the collection of the triangular facets across all the cubes forms the triangular mesh (or meshes) that defines the isosurface. The facetization pattern in each cube can be determined from the intersection topology look-up table. The processing steps for the facetization can be viewed as the isosurface assembly component of the MC.

Figure 3.37 shows the results of the surface model extracted from a volumetric model using MC. Figure 3.37 (a) is the result from direct volume rendering of the dataset, which is a female upper torso from CT scan with a size of 384*384*240. It can be seen clearly from the figure that different tissues including skin, fat, flesh and bones are rendered distinctively using transfer functions. Figures 3.37 (b) and (c) show the extracted iso-surfaces. The two data models are combined as a hybrid model as shown in Figures 3.37(d) and (e).

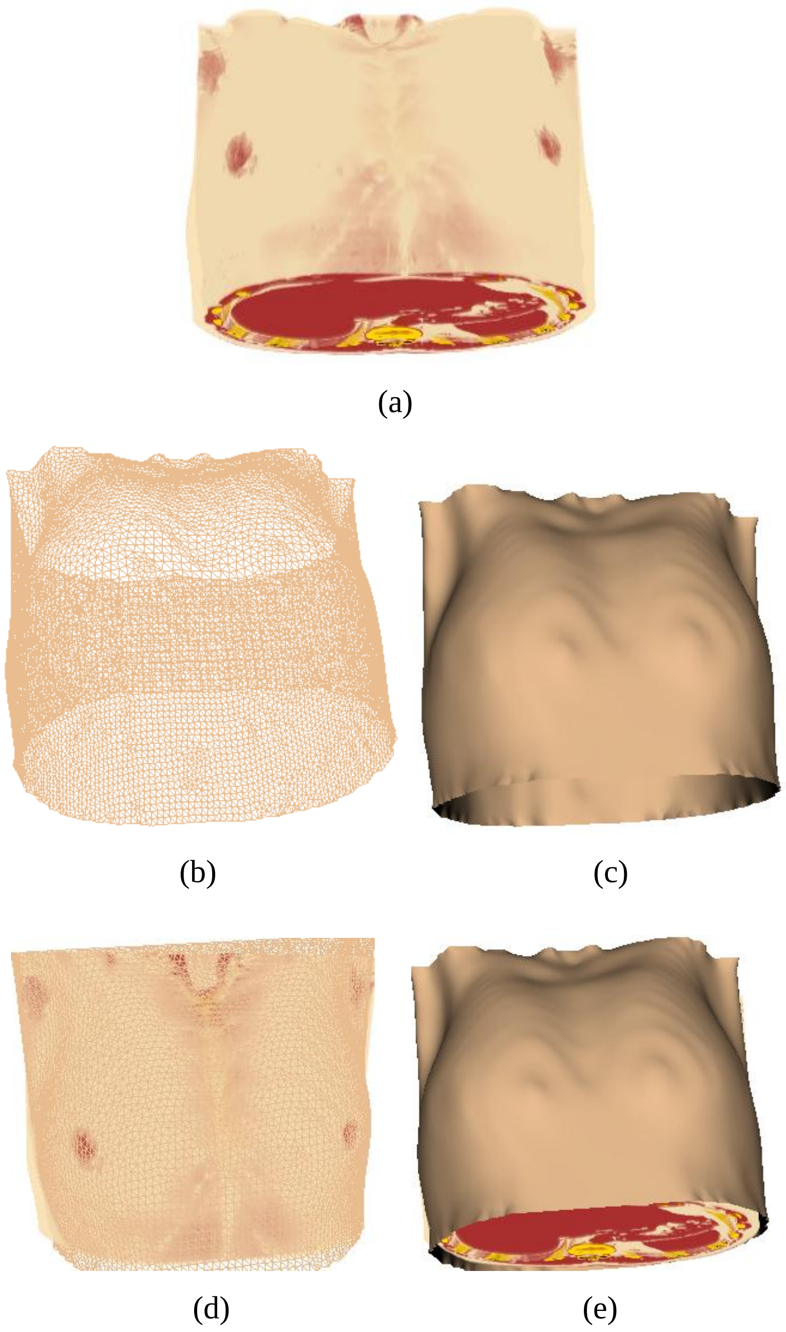
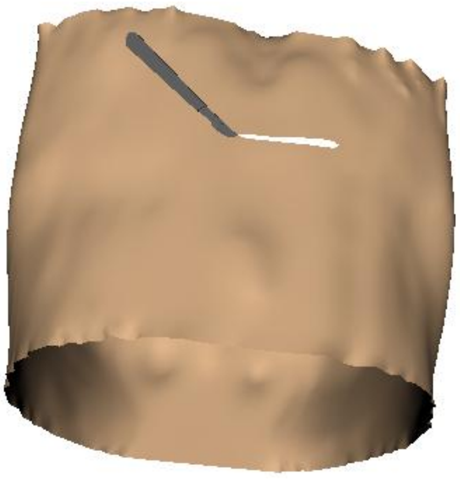


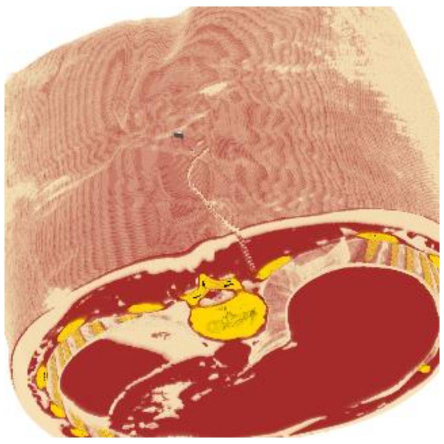
Figure 3.37: Iso-surface extraction

3.4.3 Cutting and Pulling on Hybrid Model

For a surgical training environment, the user expects immediate visual and haptic feedback while the cut is in progress, hence a simulator that supports progressive cutting is needed. We propose a method of progressive cutting which realigns the element edges while following the motion of the cutting, performed by a haptic device. The interior structure is also deformed and updated in a progressive way. The outer surface is an iso-surface extracted from the inner volumetric model in order to maintain the consistency between these data models. Direct volume rendering of interior structures provides further realism.

Progressive cutting on the surface model is illustrated in Figure 3.38(a). Free cutting with haptic interaction on the volume model is also presented as in Figure 3.38(b). The contact voxels with the haptic device are removed while the device is moving. The depth of the thin cut is dependent on the force the user exerted through the haptic device. Figures 3.38(c-e) show the results of the progressive cutting on our hybrid model. As shown in Figures 3.38 (c-e), the cutting on the outer surface model and inner volume model have been seamlessly integrated into the hybrid model progressively and simultaneously.


(a)


(b)

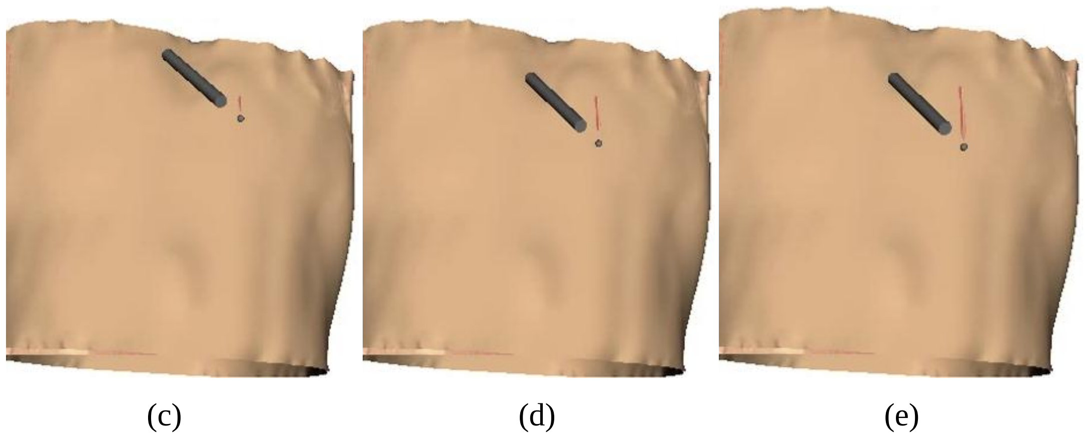


Figure 3.38: Progressive cutting on hybrid model.

If the object is simulated only by a surface mesh, the inside of the cut is not represented, i.e. it appears empty. In our system, volumetric representation is used for the interior of an anatomy object which is attached with the outer surface. We propose to generate structures in the opening as a function of the penetration depth by the haptic-controlled instrument and the width of the crack caused by the cut being pulled open. In a real surgery, after a thin cut on the object, the surgeon often uses a retractor to hold back soft tissues such as skin and muscle to open up the cut. This pulling process on the surface is achieved by real-time haptic interaction in our framework (see Figure 3.39). The width of the surface opening *D* is determined by the haptic force applied and the material properties of the organ. For example, the tissue cannot be pulled apart further than a certain constraint limit determined by the elasticity of the tissues. This pull-opening on the surface will cause the creation of a groove in the corresponding part of the volumetric model since they are integrated together. During the surface pulling process, the changing values *D* are passed to the volume model to control the width and depth of the groove in real time.

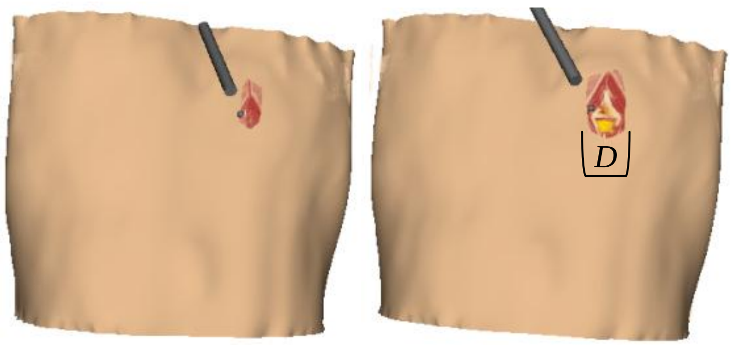


Figure 3.39: Pulling effect on the hybrid model.

3.5 Chapter Summary

Simulation of surgical cutting is one of the most challenging tasks in the development of a surgery simulator. Changes in topology during simulation render precomputed data unusable. Moreover, the process is nonlinear and the underlying physics is complex. Therefore, fully physically based simulation of surgical cutting at real-time rates on single processor machines is possibly out of reach at the present time.

In this chapter, we present a hybrid approach to the simulation of surgical cutting procedures by combing node snapping techniques on an outer surface model and groove generation on an inner volumetric model. It starts with the surface deformation initiated by haptic interaction between the device and the virtual organ surface. The outer surface model is an iso-surface extracted from the inner volumetric model by Marching Cubes Algorithm. A rupture is generated progressively based on node snapping algorithm while the haptic tool keeps moving over the surface. The incision can then be deformed and pulled for further operation. The groove is generated on the underneath volumetric model progressively to reveal the results of the cut.

Volumetric models can represent a great deal of information about the internal structures and mechanical properties of heterogeneous tissues by volume deformation and volume rendering. ChainMail algorithm could provide very fast volume deformation by constraining processing each element at most one time per deformation step. The algorithm is neither physically based nor provides any realistic tissue behaviour. It is hence nearly impossible to be used in an application which aims for realistic direct volume deformation in real time. Most ChainMail deformation systems require an extra relaxation solution for the smooth, elastic behaviours of objects. In our application, the realistic smooth deformation and cutting has been achieved on the outer surface model, and the underneath volume model is only responsible for generating the groove and presenting the interior structures and material properties. The ChainMail algorithm can hence be used for interactive deformation without the relaxation step. Furthermore, ChainMail does not care about the size of the whole datasets as only the small amount of elements in the deformed area need to be processed. Since most surgical procedures are conducted in a small area, the ChainMail algorithm well suited. A groove is generated at interactive rate on the volumetric model by the modified ChainMail algorithm. A drilling effect has also been implemented on the volume model to present the flexibility and extendibility of our model.

In our hybrid system, the outer surface model and the inner volumetric model are combined seamlessly to perform the cutting and pulling progressively and interactively. Experimental results demonstrate that our system is capable of generating realistic simulation of a surgical cut interactively. It can be included in comprehensive virtual surgical simulators to provide a realistic, safe, controllable environment for novice doctors to practice surgical operations at a low cost.

In the next chapter, the roles of haptics in all these processes will be discussed and force feedback will be integrated into the simulation system for high fidelity.

Chapter 4 Haptic Interaction

4.1 Introduction

In everyday life, the important of the sense of touch is eminent. Recent studies have shown that loss of sense of touch can be catastrophic. Skilled actions, such as using tools, holding objects, or even plain walking, may become almost impossible upon losing the sense of touch (Robles-De-La-Torre, 2006). Touch is the earliest sense developed in human embryology and is believed to be essential for good clinical practice (Fager and von Wowern, 2004). Therefore, the potential of haptic technology should not be underestimated for clinical specialities that rely on sensory input, such as minimally invasive surgery and open surgery.

The word *haptic* is of Greek origin and relates to the sense of touch – that is, the perception and manipulation of objects using the senses of touch and proprioception, which further can be divided into what is felt by the cutaneous receptors located in the skin allowing the detection of pressure, vibration, texture, heat, and pain and the kinaesthetic receptors in muscles and joints that sense the position and movements of muscles and bones (Westebring-van der Putten *et al.*, 2008).

Although haptic devices have been around for some time, realistic haptic feedback on VR simulators is difficult to achieve (Van der Meijden and Schijven, 2009), due to insufficient mechanical performance of the devices, in terms of frequency response, fidelity in fidelity in force reproduction and force resolution (Abdulmotaleb, 2012). In addition, this technology is usually an expensive add-on (Thompson *et al.*, 2011) to those VR simulation systems. Nevertheless it is still necessary to include haptic or force feedback into many VR simulators such as virtual surgery simulators as it plays an important role during surgery. The control of forces related to grasping, pressing and pulling is essential when performing surgery; the ability to control these forces may result in tissue slippage if the applied pinching forces are too small, or tissue damage if the forces are too high (Seymour, 2008). As such, the training of skills related to haptic sensations is essential (Chmarra *et al.*, 2008), and the emulation of virtual haptic feedback is an important feature of a VR surgical simulator (Basdogan *et al.*, 2004). In this chapter, we will explore the role of haptics in our hybrid cutting model, including haptics on the surface model and haptics on the volume model.

4.2 Haptics in the Surface Manipulation

A typical haptic rendering system is composed of three parts: the haptic sensory device, a visual output device, and the computer (Wang *et al.*, 2009), as shown in Figure 4.1. The system generally works as follows. Firstly, the operator applies a force to the haptic sensory device (formed by its respective sensors and mechanical structures). Next, based on the amount of force applied and the pre-defined model of the object of interest, the computer computes the feedback force and the deformation of the object, and sends the data to the haptic sensory and visual output devices. The operator can simultaneously feel the feedback force from the object and see the object deformation through the visual output device.

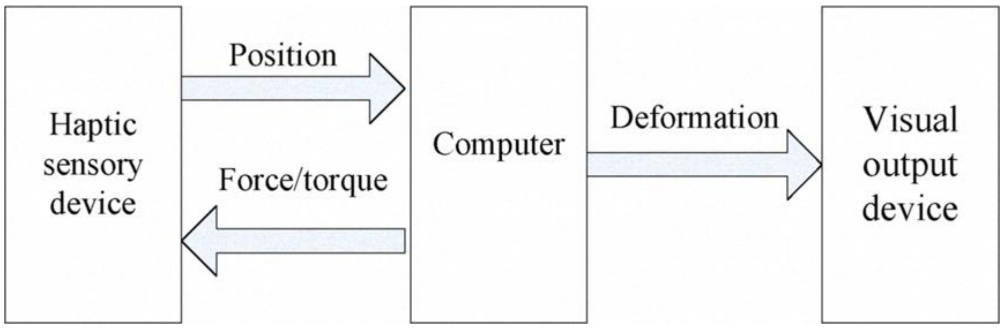


Figure 4.1: The composition of a haptic rendering system for virtual reality.

The computer is the control centre of the system. It needs to implement mainly two algorithms: collision detection and collision response. When the operator moves the probe, the computer estimates the new position and orientation of the probe and test if the probe collides with the virtual object. In case of a collision, the force between the probe and the object is computed, and the computed data are then fed back to the operator through the haptic sensory device as a collision response. For a smooth rendering of feedback force to the operator, the force needs to be computed at a frequency around 1 KHz, while the visual image should be updated at around 30 Hz.

Fast and accurate collision detection is crucial in any VR-based surgery simulation system. In collision detection, one needs to ensure that there has indeed been a contact between the surgical tool and the objects in the scene. The bounding-box based algorithms prove to be the most efficient data structures for collision detection.

The main principle is to represent the object by a set of regularly shaped bounding boxes. The tree-structured hierarchy is constructed recursively to approximate the object as much as possible. Currently, there are various kinds of bounding boxes developed for many applications in computer graphics and visualization (Teschner *et al.*, 2005), such as oriented bounding box (OBB), discrete orientation polytope (k-DOPs), bounding sphere, and axis-aligned bounding box (AABB) (Chen *et al.*, 2012), etc.

Generally, the collision model in haptic rendering defines the graphical description of the surgical tools and the nature of tool – tissue interactions. The virtual tool may be modelled as a point, a ray, or as a 3D object. However, no matter how the virtual tool is modelled, the collision detection generally only considers the collision between the tip of the probe, typically referred to as the Haptic Interface Point (HIP), and the virtual object.

Once collision has been detected, the interaction of the surgical tool with the virtual organ model is computed during collision response. The deformed organ model is displayed on the computer screen. The reaction forces are fed back to the user through the haptic interface device(s). For force calculation, the depth of probe into the object, i.e., the distance between the HIP and the surface of the object, is first calculated. The applied force is then computed through the Hooke's Law of elasticity F = -*k*x. This is a simplified calculation method. In real life computations, damping and friction also need to be considered to ensure system stability and the validity of the haptic sensory data.

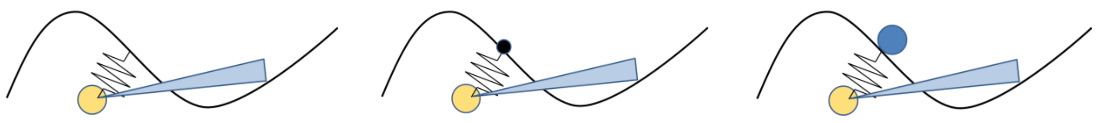
4.2.1 Surface Haptics

Algorithms for haptic interaction with explicit surface information are designed to generate a force feedback when the haptic probe comes in contact with the surface. Since a force feedback device (impedance control) is incapable of explicitly controlling the position of the haptic probe, the surface simulating control system must allow the probe to penetrate surfaces. When a surface is penetrated, however, a force is applied to stop the probe from penetrating further. This is the basic principle of surface rendering which is common to all algorithms for impedance-based haptic feedback.

The penetration of surfaces is not as serious an issue as it might seem. Since the kinesthetic sense of touch has a low resolution at low frequencies, the displacement is not as prominently perceived through touch as it is through vision. Thus, the impression of penetration can be reduced by giving the visual impression of the haptic instrument not penetrating the surface. This way the increasing resistance when applying increasing force and penetrating the surface further is rather perceived as an increasing force applied to the surface.

The first developed approach used for force feedback from geometrical surfaces applies a force that pulls the haptic probe towards the closest point on the closest polygon of the object boundary, as shown in Figure 4.2(a). The force strength is made proportional to the penetration depth, as if a spring was connected between the probe and the surface. This gives a feedback force that is the effect of a "penalty" from the penetration of the polygon surface, which gives it the name *penalty method*.

The penalty method suffers from artefacts that make it impractical in real applications. Since the approach represents a static control it has, at the time the feedback is calculated, no memory of which surface was previously palpated. Because of this, the algorithm can suddenly start to treat another surface that is at this instant closer to the probe than the one currently palpated. This gives rise to such artefacts as popping-through of thin objects when the opposite side of the object suddenly becomes closer to the probe than the first palpated side, and discontinuities around edges and corners. To remove these artefacts the system needs a memory of what part of the palpated object was touched the last time the haptic feedback was estimated. This memory is implemented through a virtual object that is left on the surface that the probe penetrates. Each time the feedback is calculated for a haptic frame, the system now knows the previous surface position.



(a) The *penalty method* generates a force towards the closest point in the closest polygon.

(b) The god-object method uses a point on the polygon surface to serve as memory of which surface is currently palpated.

(c) The proxy method uses a finite sized sphere as proxy for the haptic probe to avoid the proxy to slip through inter-polygonal cracks.

Figure 4.2: The three most common approaches for generating haptic feedback from polygonal surface data.

The first implementations following this approach used a single surface point called a *god-object*, as shown in Figure 4.2(b). Using a single point as memory for interaction, however, has some disadvantages. Numerical errors in the estimation of triangle surfaces sometimes leave small gaps between the triangles composing the surface of an object, and such gaps have proved large enough for the god-object. Thus, the god-object could fall through object surfaces. The god-object method was refined by Ruspini *et al.* (1997) to avoid the need for the explicit topology information by the introduction of a finite-sized spherical *proxy* object, as shown in Figure 4.2(c).

The *proxy* is an internal representation of the haptic probe. It is fully controlled by the surface simulation algorithm and can be constrained by surfaces in a stable manner. The force feedback is then calculated by simulating the coupling through a virtual spring damper,

$$\vec{f}_{fb} = -k(\vec{x}_{probe} - \vec{x}_{proxy}) - D(\vec{\upsilon}_{probe} - \vec{\upsilon}_{proxy}) \tag{4.1}$$

where k is the stiffness of the coupling and D is the dampening term. In free space the proxy is automatically moved to the position of the probe and no feedback is generated through the virtual coupling. When an object is penetrated by the probe, the proxy is moved over the surface towards the probe. By modulating the movement of the proxy over the surface, other effects can be generated, such as friction, texture, haptic shading and even bump-maps. Because both the god-object method and the proxy-based method let surfaces constrain the movements of the proxy object, they are sometimes also referred to as *constraint-based* method.

4.2.2 Haptics in Surface Deformation before Rupture

Generally each object within the virtual environment has a tree of hierarchical bounding volumes. Deformable objects typically use bounding spheres because the update of bounding volume is simple to implement once the object undergoes deformation.

Figure 4.3: 3D model of scalpel.

The most basic cutting instrument is a straight, rigid object that has one sharp surface, such as a scalpel in Figure 4.3. When modelling this object, we assign the tip of the scalpel as the HIP, which means the collision detection will be based on the position of the surgical tool tip and the soft tissue. Once a collision is detected, the force feedback is calculated and sent back to the user. As long as the force is smaller than the yield limit of the material being simulated, the surface will start the deformation based on PAFF. Since in this stage the movement of the cutting instrument is typically limited to pushing instead of moving or sliding, the force calculation could be simplified to Hooke's Law of elasticity and no damping and friction need to be considered, as shown in Figure 4.4. The force produced is a linear spring-damper force between the proxy position and probe position, i.e. F = stiffness * (proxy_pos - probe_pos).

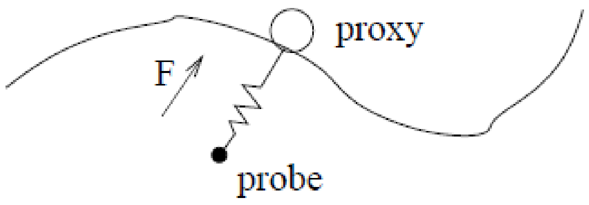


Figure 4.4: Force calculation for surface deformation.

4.2.3 Haptics in Progressive Surface Mesh Cutting

When the haptic force has been applied on a deformable object, the surface of the object displaces until the applied force becomes larger than the yield limit of the material being simulated. When the breaking strength of the material is reached, further haptic tool motion causes the object to be cut.

During the progressive cutting, collision between the blade and the virtual organ has been constantly detected. Intersection points between the HIP and the underlying polygon edges are calculated for the node snapping algorithm. When cutting with a knife or scalpel, the shape of the blade means that the friction is not the same in every direction. For example, it is easy to cut along the direction of the blade, but if we try to move the knife sideways or backwards it is much harder to move. Thus the force feedback calculation is based on a frictional surface that is dependent on the angle of the haptic device.

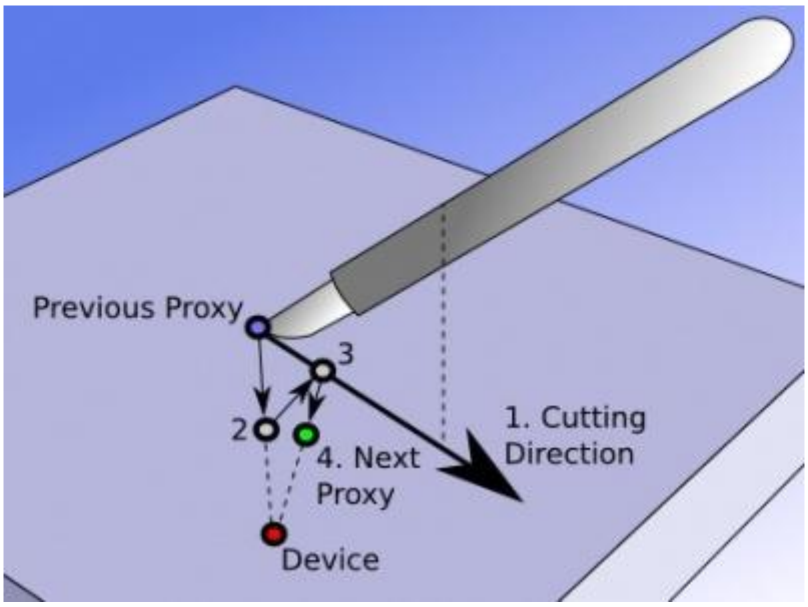


Figure 4.5: Force calculation for progressive surface cutting.

Figure 4.5 illustrates how to calculate the proxy movement for our surface. The final proxy movement will be a vector from the previous proxy position (the blue dot), to the next proxy position (green dot). The calculation of the total proxy movement for our directional frictional effect consists of four steps:

1. The cutting direction is determined. To do this a unit vector is projected onto the surface representing the angle of the scalpel/device.

2. The proxy movement allowed by the friction surface is worked out which represents friction along the cutting direction. The movement of the proxy after performing this step is labelled as 2 in Figure 4.5. Note that the proxy moves from the previous proxy position (blue dot) towards the device position (red dot). If the friction parameters were zero (i.e., no friction), the proxy movement would take the

proxy all the way along the dotted line, to a point on the surface, closest to the device position.

3. In step 2 the movement of the proxy permissible by friction along the line of cutting is calculated. However the movement is always towards the device position, and is not constrained by the direction of cutting. So the next step is to project the proxy movement onto the cutting direction, effectively restricting movement to the line of cutting. The resulting projection is labelled as 3 in Figure 4.5. Apart from constraining movement to the cutting line, the movement should also be restricted to the direction of cutting, i.e., we do not want to allow backwards movement. Therefore, the angle between the proxy movement and the cutting direction is checked before the projection. If the proxy movement is found to be backwards, it would be nulled.

4. Now the movement is constrained to the cutting direction and can be affected by friction. However, currently there is no possibility to allow any movement away from the cutting direction. It is desirable to permit some movement away from the cutting line. Thus the proxy movement is calculated from the proxy position on the cutting line, which was calculated previously (labelled as 3 in Figure 4.5) towards the device position (shown as the red dot in Figure 4.5). This allows the next proxy position (green dot) to slip away from the cutting direction by an amount that can be controlled by adjusting the friction parameters.

The directional friction force is fed back to the users so that they can feel the different level of friction when they attempt to move the scalpel over the surface, especially by holding the scalpel at different angles.

4.2.4 Haptics in Deformation of Surface Incision

In a real surgery, retractors or spatulas are used to separate the cut by holding back soft tissues such as skin and muscle while the surgeon attempts to make further cuts or to focus on the region of interest. In our application, this pulling process is achieved by real-time haptic interaction. In this stage, the haptic device is modelled as a retractor. For easy manipulation, we implemented a *magnetic effect* on the virtual organ surface where the proxy is attracted to the surface, and forces are generated in order to keep the proxy on the surface. The surface has a parameter called *snapDistance* to define a distance from the surface within which forces are generated to pull the proxy towards the surface. If the device is pulled outside the distance from the surface it will be freed from the magnetic attraction. To deform the surface incision, we hold the haptic device near one of the cutting sides. If the device is within the snapDistance of the surface, the proxy will be attracted to the closest point. This closest point is used as a *control point* for deformation and the incision is opened up along the direction of the haptic movement until some boundary has been reached. For example, the tissue cannot be pulled apart further than a certain constrained limit determined by the elasticity of the tissues. Figure 4.6 shows different incision opening controlled by haptic interface. Our algorithm can be easily extended to simulate other surgical operations such as suturing by knotting the two sides of the incision back together.

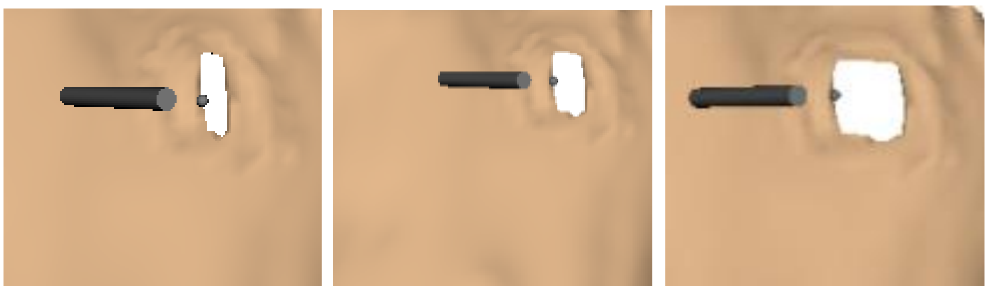


Figure 4.6: Different widths of surface cut opening controlled by haptic device.

4.3 Haptics on Volumetric Model

When haptic interaction was made available in computer graphics applications, it was generally based on the notion of surfaces. Haptic feedback is generated as a response to touching geometrical representations of surfaces in the virtual environment, and the feedback is generally perceived as a surface. Different algorithms for haptic interaction with surfaces all strive towards a common goal – more stable and more correct or realistic surface feedback. Haptic interaction with volumetric data is different. In volumetric data there are no explicit surfaces. Some volumetric datasets contain data that could be interpreted as surfaces, but not all of them. Since the volumetric data cannot be directly interpreted in a straightforward haptic form, haptic representation of the contents must be generated in a selected or designed manner. The term *haptic mode* was suggested by Pao and Lawrence (1998) to describe the unique haptic impression in interaction with volumetric data. A haptic mode is a distinct haptic representation of data that provides a unique connection between data and representation. Thus, different haptic modes applied to the same data may

1. provide haptic representations of different properties of the volumetric data
2. provide different haptic effects representing the same property of the data

Volumetric datasets may contain different types of attribute data, both with respect to the dimensionality (scalar, vector or tensor), and with respect to what the values in the dataset represent. A vector volume may, for example, represent air or fluid flow information generated through CFD, or the strength and orientation of a magnetic field. While a certain haptic mode may be compatible with scalar data, it might not work with vector data. Similarly, a mode compatible with vector data but designed specifically for intuitive iteration with fluid flow data might not be appropriate for exploration of a magnetic field, even if the data is compatible. The mode could be counter intuitive or simply not convey the most important properties of the data.

Chen and Sun (2002) created a system for sculpting both synthetic volume data and data obtained from CT, MRI and Ultrasound sources. The direct haptic rendering approach utilized an intermediate representation of the volume data (Chen *et al.*, 2000). The intermediate representation approach to haptic rendering was inspired from its use in rendering geometric models (Mark *et al.*, 1996). The sculpting tools developed by Chen and Sun were treated as volumes allowing each position in the tool volume to interact with the object volume data. They simulated a variety of sculpting effects including melting, burning, peeling and painting.

When interacting with the volume data directly, an approach is required to provide stiff and stable contacts in a similar fashion to the rendering achieved with geometric representations. This is not easily accomplished when using the techniques based on mapping volume data directly to forces and torques. One strategy is to use a proxy constrained by the volume data instead of utilizing an intermediate representation as in the previous example (Palmerius, 2007). Lundin *et al.* (2002) presented an approach aimed at creating natural haptic feedback from density data with solid content (CT scans). To update the movements of the proxy point, the vector between the proxy and the Haptic Interface Point (HIP) was split into components: one along the gradient vector (f_n) and the other perpendicular to it (f_t). The proxy could then be moved in small increments along f_t. Material properties such as friction, viscosity and surface penetrability could be controlled by varying how the proxy position was updated. Palmerius (2007) developed an efficient volume rendering technique to encompass a constraint based approach with a numerical solver and importantly a fast analytical solver. The proxy position is updated by balancing the virtual coupler force, *f*, against the sum of the forces from the constraints, F_i. The constraints are represented by points, lines and planes. The balancing is achieved by minimizing the residual term, ε, in the following equation:

$$\vec{\varepsilon} = -\vec{f}(\vec{x}_{proxy}) + \sum_i \vec{F}_i(\vec{x}_{proxy}) \qquad (4.2)$$

By modifying the effects of the constraints in the above equation, different modes of volume exploration can take place such as surface-like feedback and 3D friction. Linear combinations of the constraint effects can be used to obtain the combined residual term. An analytical solver may then be used to balance the equation and hence find the position of the proxy. The analytical solver is attempted first for situations where the constraints are orthogonal; however, if this fails a numerical solver is utilized. This combination of techniques is available in the open source software entitled Volume Haptics Toolkit (VHTK).

4.3.1 Haptics in Volume Deformation

During the groove generation, the haptic device is used as an interaction tool to pull the two sides of the cut apart. The voxel nearest to the haptic stylus is chosen to be the initial moved element and the deformation starts afterwards.

VHTK is employed to handle the haptic interactions of the volumetric data in our system. VHTK provides a number of haptic modes, each giving a unique haptic representation of the volumetric data. Haptic modes in VHTK are implemented by controlling parameters of *haptic primitives* as functions of the volumetric data that the haptic mode is representing. There are currently four primitives available in VHTK, one constraint for each dimensionality – point (3D), line (2D) and plane (1D) primitives and one force primitive. Each primitive has a strength parameter, which specifies the strength of the feedback from the primitive. The force, line and plane primitives also have a direction parameter, which can be used to represent some vector feature in a haptic mode. A haptic mode may use one or several haptic primitives to generate its specific haptic effect. The lowest-level system of VHTK prompts the haptic modes for the instant set of haptic primitives to momentarily represent the haptic feedback at a rate of about 1 KHz. The haptic mode then reads off the local data properties and sets up one or more primitives to reflect these properties. These primitives are then returned to the system to calculate the haptic feedback for that time-frame.

Figure 4.7 and Figure 4.8 show how the incisions on volumetric model and the hybrid model are widened up and pulled away by the haptic device respectively.

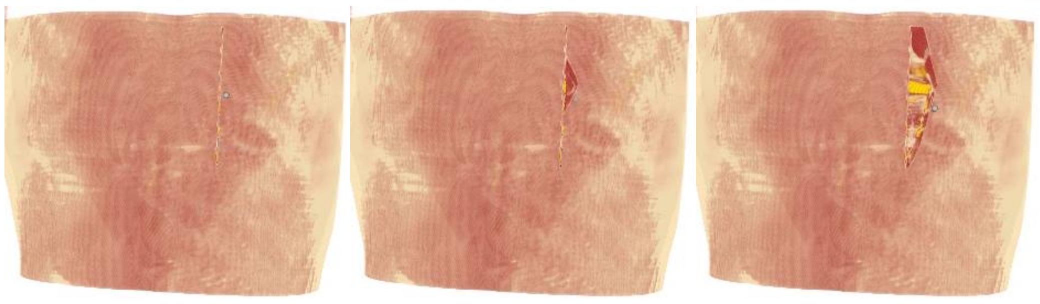


Figure 4.7: Different widths of volume cut opening controlled by haptic device.

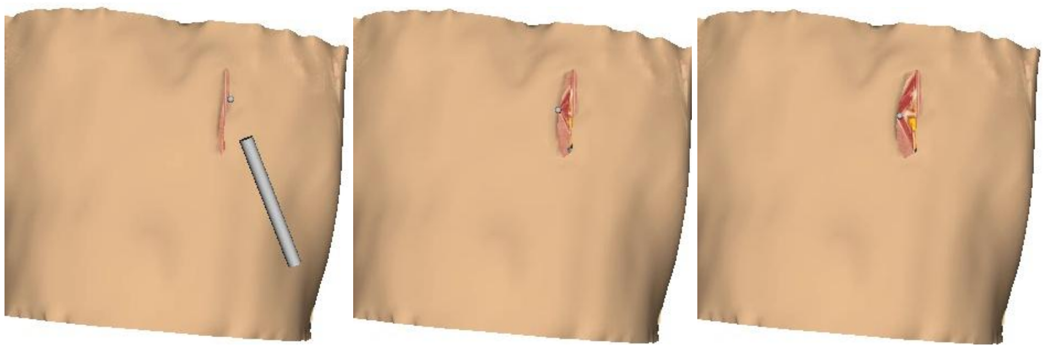


Figure 4.8: Different widths of hybrid cut opening controlled by haptic device.

4.4 Chapter Summary

An important aspect of VR surgical simulators is the reproduction of the challenges a surgeon encounters during surgery – challenges related to vision and touch. In surgery, haptic or force feedback refers to the sense of touch that a surgeon experiences – both consciously and unconsciously – while performing surgery. Haptics provide sensation to numerous surgical procedures, varying from structure to structure and depending on type of force applied, and relates to tissue damage, straightness of suturing, and task completion time. In this chapter, haptic interactions on surface model and volume model have been discussed which could provide high fidelity for the simulation system.

In next chapter, a fluid model is integrated with the surgery cutting for simulating the bleeding effects in real time to greatly enhance the realism of surgical simulations.

Chapter 5 Dynamic Fluid Visualization for Bleeding Simulation

5.1 Introduction

Bleeding is an inevitable part of most surgical procedures. A cautery tool is often used to divide tissue with minimal blood loss. Sometimes, the tissue may also bleed if coagulation is not efficient. Surgical errors may lead to excessive bleeding and possibly even death. The presence of blood or haemorrhages in the operating field is a fundamental challenge in surgery since the visibility of key anatomical structures is impeded. Therefore, intraoperative management of bleeding is a critical skill all surgeons must possess. Recent surgical literature prescribes the use of such cues as important assessment factors in surgical education, e.g. control of bleeding has been measured as a parameter while assessing the surgical skill of medical trainees and was found to correlate well with skill level (Bemelman, 2009). These and other studies establish the need to include bleeding as features to be integrated into surgical simulators for skill training.

The existence of fluid during surgeries can sometimes obscure a surgeon's view and make the operation more difficult. For some surgical procedures, saline fluid is injected or flooded into the operation area for cleaning purpose. Unpredictable blood splashing might also be resulted from incisions on organs. Different kinds of fluid are commonly sucked away or drained from the operation area from time to time. Dynamic fluid simulation can therefore enhance the realism in surgical simulation. Without fluid, surgical simulations appear dry and clean, which would be unrealistic for most surgical procedures.

Although blood provides important visual cues which a surgeon-in-training must be able to recognize to make time-critical decisions, it is, however, very challenging to create a safe and realistic learning environment for the acquisition of such skills. Most existing virtual realities surgery simulators tend to avoid including dynamic fluid simulations in their system. This is due to the fact that real time fluid visualization is computationally expensive. The calculation may slow down rendering and impede force feedback interactions during simulations. VR-based surgical simulators must be *interactive*, i.e. the computations must be performed in *real time*. For haptic (touch)-enabled systems, response forces to the users must be updated at 1 KHz, while visual updates must be maintained at 30 Hz for monocular and 60 Hz for stereo rendering. This is a challenging task, as a fully functional surgical simulator typically performs multiple tasks concurrently, including realistic rendering of the surgical scene, collision detection, computation of tissue response, accurate tool–tissue interactions and haptic feedback. Incorporating all of these requires significant computational power. Since the various components of the surgical simulation system run at different frequencies, they must also be synchronized for data integrity and consistency. Therefore, simulation of bleeding is mostly either ignored or simulated using highly simplified techniques, since other computationally intensive processes compete for the available CPU resources.

With the availability of high performance graphic hardware, the improvement of visual quality and computational accelerations become easier to achieve. Modern GPUs are highly optimized data-parallel streaming processors. The major innovation in recent years was the replacement of the traditional fixed-function pipeline by a programmable pipeline, which allows the programmer to upload user-written microprograms to be executed very efficiently. By outsourcing the computations and rendering of fluid to the GPU, the CPU could be freed up for other time-critical tasks and the whole system could still achieve real-time performance.

Much has been written on algorithms for computational fluid dynamics. Stam (1999) described an algorithm for a fast Navier-Stokes equation solver that provides stable results suitable for graphics applications. This solver has been further developed by many seeking to enhance both the quality and performance. One example is the system developed by Crane *et al.* (2007), which provides a modification of the Stam algorithm that uses NVIDIA’s shading language Cg and frame buffer objects in OpenGL to run the solver using programmable graphics hardware.

As discussed in Chapter 2, real-time techniques for the generation of bleeding effects may be categorized into two major groups: non-physical, and physical. The nonphysical approaches are based on rendering techniques, where the focus is to create visually appealing rather than physically accurate results. The physical approaches offer the most accurate simulation of the underlying physics, but they are computationally demanding, thus not suitable for VR-based interactive applications such as virtual surgery. Given that the major goal of bleeding simulation in virtual simulators is to provide visual realism without introducing significant additional load on the system, non-physical methods appear to be a sensible choice for present day surgical simulators, as many of the physical approaches require too much computation to be integrated into real-time applications. This is critical for current surgical simulation systems because of the enormous computational load already placed on the CPU from simulating deformations, collisions and response characteristics. We have developed a texture based approach that uses pixel and vertex shader level techniques in the GPU to utilize its parallel many-core architecture (Owens *et al.*, 2007) for both the computation and rendering of the fluid. It is a non-physical simulation but capable of simulating the effect of liquid flowing along a surface and providing a realistic visual clue in surgical simulations.

In the next section, a short introduction of both Lagrangian based and Eulerian based fluid simulation approaches is given. Section 5.3 details our fluid implementation based on cellular automata on programmable graphics hardware; and Section 5.4 summaries this chapter.

5.2 Basic Fluid Dynamics

Computational fluid dynamics (CFD) has been an important tool for scientists and engineers ever since the performance of computers reached useful levels during the 1960's. Many technological feats, such as going to the moon and back, modern jet fighter aircraft and nuclear submarines would have been more or less impossible to achieve without the help of CFD (Anderson, 2005). It is a huge subject that has received a great deal of attention and research funding during the last decades, partly due to the obvious military applications. Here we will just scratch the surface and review a couple of the most fundamental equations relevant to fluid simulation in the context of computer graphics and visualization.

5.2.1 The Navier-Stokes Equation

The fundamental equations governing the motion of a fluid are the Navier-Stokes equations, derived independently by the French engineer Claude Navier and the Irish mathematician George Stokes in the first half of the nineteenth century. These equations can take many forms depending on the assumptions made. The fluid is assumed to be incompressible and Newtonian (Anderson, 1995), and thus the Navier-Stokes equations take the following form

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} + \mu\nabla^2\mathbf{v} - \frac{1}{\rho}\nabla p + \mathbf{f} \qquad (5.1)$$

Where $\mathbf{v}$ is the fluid velocity, ρ is the density, p is the pressure, μ is the kinematic viscosity and $\mathbf{f}$ is the external body force. The first term on the right, $-(\mathbf{v} \cdot \nabla)\mathbf{v}$ is called the *convection* term. It basically represents the change of velocity of a fluid particle caused when the particle moves from one region of the velocity field into another region with different velocities. It can be seen as a "transport of velocity" by the velocity field itself. The second term is the *viscosity* term, representing the internal friction and normal stresses generated between fluid particles as they move in relation to each other. The third term is the *pressure* term, stating that particles are pushed in the direction of the negative pressure gradient. The last term represents body forces, such as gravity or a magnetic field, acting directly on the matter constituting the particle.

5.2.2 The Continuity Equation

To describe the motion of a physical fluid the Navier-Stokes equations need to be complemented with an equation assuring that no mass is created or destroyed in the process. This fact is described by the *continuity equation,*

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\mathbf{v}) = 0 \qquad (5.2)$$

It states that the rate of density change of an infinitesimal fluid element equals the total amount of mass per volume entering and leaving the volume occupied by the element. In other words, mass is conserved.

5.2.3 Lagrangian Method

The Lagrangian approach treats the continuum as a particle system. Each point in the fluid is labelled as a separate particle, with a position x and a velocity u. with the Lagrangian viewpoint, the incompressible Navier-Stokes equations are derived as

$$\nabla\mathbf{v} = 0 \qquad (5.3)$$

$$\frac{\partial \mathbf{v}}{\partial t} = \mu \nabla^2 \mathbf{v} - \frac{1}{\rho} \nabla p + \mathbf{f} \qquad (5.4)$$

The left hand side of the momentum equation can be interpreted as the acceleration of a particle while the right hand side is the net force exerted. The equations have been simplified assuming that the fluid is incompressible since most visually appealing fluid effects in real have little compressibility (Bridson and Müller-Fischer, 2007).

Reeves (1983) introduced the idea of particle systems which are then widely used to model the deformable bodies, clothes and other chaotic phenomena. The particle system is an irregular discretization of the continuum. In order to solve the Navier-Stokes equations, the gradient operator ∇ and laplacian operator ∇^2should be well defined under such as irregular discretization. Monaghan (1992) introduced the smoothed particle hydrodynamics (SPH) method into the computer graphics community to address this issue. It defines a smoothing kernel to interpolate the physical properties (velocities, densities, etc.) at an arbitrary position from the neighbouring particles. The fluid is represented by a set of particles $i \in [1 \dots N]$ with positions x_i, masses m_i and additional attributes A_i (velocities, densities, etc.). SPH defines how to compute a smooth continuous field *A(x)* from the discrete attribute values A_i sampled at particle locations x_i as

$$A(x) = \sum_i m_i \frac{A_i}{\rho_i} W(x - x_i, h) \qquad (5.5)$$

The kernel function $W(r,h)$ is typically a smooth, radial symmetric, normalized function with finite support. For example, in Müller's work, the kernel was designed as

$$W(r,h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |r|^2) & 0 \le |r| \le h \\ 0, & otherwise \end{cases} \qquad (5.6)$$

The gradient and laplacian of the smoothed attribute function *A(x)* are

$$\nabla A(x) = \sum_i m_i \frac{A_i}{\rho_i} \nabla W(x - x_i, h) \qquad (5.7)$$

$$\nabla^2 A(x) = \sum_i m_i \frac{A_i}{\rho_i} \nabla^2 W(x - x_i, h) \tag{5.8}$$

Consequently, the right hand side of momentum equation can be easily discretized with the above definitions.

This meshless method of using Lagrangian particles can operate more easily with irregular boundaries, between multiple fluids interaction and generally requires less computational resources. Though Lagrangian approach has been widely used in many interactive applications, due to the difficulties in surface reconstruction and rendering, particle-based method have not yet demonstrated the same level of realism as its grid-based counterparts.

5.2.4 Euler Method

The Eulerian approach follows another strategy. Instead of treating the fluid as flowing particles and then tracking each particle, it looks at fixed points in space and sees how the fluid quantities (including densities, temperatures and velocities) measured at those points change with time. Thus, the whole fluid region is modelled as fields of fluid quantities. For a specific time and a given position, there exists a group of values to represent the fluid state. For instance, the vector field $v(x,y,z,t)$ is to characterize the velocities and the scalar field $p(x,y,z,t)$ it to measure the pressure inside the fluid. The incompressible Navier-Stokes equations have the following form with Eulerian viewpoint:

$$\nabla \mathbf{v} = 0 \tag{5.9}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} + \mu \nabla^2 \mathbf{v} - \frac{1}{\rho} \nabla p + \mathbf{f} \tag{5.10}$$

In Eulerian methods, the above equations are discretized with the grids. The finite difference methods are used to solve the equations numerically. There are two ways to store the fluid quantities on the grid. The most popular way is to store the scalars, such as pressures, level set values and temperatures at the centre of each grid and store the vectors, such as velocities at the faces of each grid cell. This staggered configuration of MAC grid was first presented by Harlow and Welch (1965), which benefited from its unbiased and second order accurate central difference scheme. Most of the state-of-art simulations adopted the staggered grid. Another way is to store all the quantities at the node of each grid cell, such as in Stam's paper (2003). The advantage is simplicity. There is no need to handle different variables differently. Interpolations are simplified significantly as well.

At the early stages of physically-based fluid animation, researchers did not tackle the sophisticated Navier-Stokes equations directly. On the contrary, they made several assumptions and reduced the governing equation to the ware equation (Kass and Miller, 1990) or the shallow water equations. The height field was used to represent the water surface. Although this algorithm was quite simple and efficient, a myriad of interesting fluid phenomena, such as overturning wares, sprays and splashes, could not be captured. Foster and Metaxas's work (1996) was the first example that solved the full 3D Navier-Stokes equations to animate fluids. Stam (1999) improved it, achieving the unconditionally numerical stability by introducing the semi-Lagrangian method for the convection term and implicit solver for the viscosity and pressure terms. It became the standard framework to implement fluid animation codes. The Eulerian method can be divided into four steps, from the initial velocity $\boldsymbol{v}_0(x)$ to the resultant velocity $\boldsymbol{v}_4(x)$ after one time step:

$$\mathrm{v}_0(x) \xrightarrow{\overbrace{\hphantom{xxxxx}}^{\text{Add force}}} \mathrm{v}_1(x) \xrightarrow{\overbrace{\hphantom{xxxxx}}^{\text{Advect}}} \mathrm{v}_2(x) \xrightarrow{\overbrace{\hphantom{xxxxx}}^{\text{Diffuse}}} \mathrm{v}_3(x) \xrightarrow{\overbrace{\hphantom{xxxxx}}^{\text{Project}}} \mathrm{v}_4(x)$$

The four sub-steps are

Body forces: $v_1(x) = v_0(x) + \Delta t f$ (5.11)

Advection: $v_2(x) = v_1(p(x, -\Delta t))$ (5.12)

Diffuse: $(I - \mu\Delta t \nabla^2) v_3(x) = v_2(x)$ (5.13)

Projection: $\nabla^2 p(x) = \dfrac{\rho}{\Delta t} \nabla \cdot v_3(x)$ (5.14)

$$v_4(x) = v_3(x) - \frac{\Delta t}{\rho} \nabla p \qquad (5.15)$$

Combining the Eulerian method and level set based surface tracking algorithms (Foster and Fedkiw, 2001) has produced stunning results, simulating various interesting fluid phenomena such as smoke, water, fire, droplets, non-Newtonian flow, bubbles, etc. In general, these techniques have progressed to the point where fluid phenomena can be modelled so realistically that a naïve viewer may have difficulties in telling reality from simulated footage.

Although some speedup algorithms have been proposed (Carlson *et al.*, 2004; Qin *et al.*, 2007), most of the physically-based methods are still not suitable for real-time and interactive surgical simulation mostly due to the fact that they are computationally very demanding and the cost increases dramatically with increase in grid size or the number of particles, especially when complicated models are used for acceptable visualization realism. In the next section we will discuss in details our non-physical method for bleeding simulation based on Cellular Automata model.

5.3 Fluid Simulation based on Cellular Automata

In our system, the cutting simulation is augmented with real-time fluid dynamic visualization. As physically based approaches for fluid simulation are generally computationally expensive due to the calculation of the Navier-Stokes equations, we propose an alternative way for fluid simulation. Rather than using physically-based methods, a texture based fluid simulation is used to generate the bleeding effect. Although it is not numerically accurate, it is able to provide a realistic visual cue in surgical simulations. The computations and rendering of the fluid are both performed on GPU so that the CPU could focus on other time critical tasks such as progressive cuttings and real time deformations. Our method for bleeding does not depend on the size and the geometrical shape of the organ or its associated textures, and can be applied in any complex simulation scene without significant reduction in performance. Unlike previous techniques (Halic *et al.*, 2010; Rianto and Li, 2010) where the fluid only occurs on the static surface, our fluid can start from the cutting area on the outer surface, and propagate into the inner volumetric model during the process of groove generation so that the whole surgical scene looks more realistic.

Our simulation is done by a cellular automaton residing on the surface of the object. A cellular automaton is a discrete model for the description of physical dynamic systems regulated only by local laws. Typically it consists of a regular grid of cells, each in one of a finite number of states or properties, such as “on” and “off”. For each cell, a set of cells called its *neighbourhood* (usually including the cell itself) is identified. An initial state is selected by assigning a state for each cell. The system evolves in time according to some fixed rule (generally, a mathematical function) that determines the new state of each cell based on the current state of the cell and the states of the cells in its neighbourhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously.

A common approach in fluid simulation relies on top down viewpoints that uses 2D/3D mesh based techniques in conjunction with fluid equations (Chentanez *et al.*, 2006). However, a common challenge is the cost of the computational animation of the solid-fluid interaction. Recently, the ability of CA algorithms (with simple rules) to simulate the complexity of physical models has been investigated in many studies in order to improve dramatically the computation time of 2D surface fluid models (Coulthard *et al.*, 2007; Liu and Pender, 2010). These methods using bottom up models for surface fluid flow simulation and are cheaper than the traditional ones for fluid simulation, because there is no need to solve Partial Differential Equations (PDEs) but to compute the non-iterative operation of the CA rules. Furthermore, CA algorithms are well suited to be executed in parallel in modern high performance hardware, thus obtaining a very fast speed of computation. It has been applied for fluid simulation in VR applications such as computer games (Judice *et al.*, 2008), and digital terrain models (Coutinho *et al.*, 2008; Cirbus and Podhoranyi, 2011). To the best of our knowledge, there is no other study to date that uses this method in surgery simulation for bleeding effect.

5.3.1 Overview of Cellular Automaton

The standard approach to simulating natural phenomena is to solve equations that describe their global behaviour. While the results are typically numerically and visually accurate, most of such simulations require too much computation (or small lattice sizes) to be integrated into interactive graphics applications such as virtual surgery. Cellular Automata models (Schiff, 2011) do not solve for the global behaviour of a phenomenon. Rather they model the behaviour by a number of very simple local operations. When aggregated, these local operations produce a visually accurate approximation for the desired global behaviour.

Cellular automata could provide an artificial, mathematical model of a dynamic system, and be employed for the purpose of simulating real systems, by representing the state of a dynamic system as continuous values on a discrete lattice.

A cellular automaton refers to a model consisting of the following components:

- A grid composed of cells.
- A set of ingredients.
- A set of local rules governing the behaviours of the ingredients.
- Specified initial conditions.

Once the above components of the model are defined, a simulation can be carried out. In the simulation the system evolve via a series of discrete time-steps, or iterations, in which the model rules are applied to all the ingredients of the system, and the configuration of the system is updated accordingly.

The grid in a CA model may contain a single cell, or more commonly a large collection of, possibly as many as 100,000 or more, cells. In principle, the grid itself might be one-, two-, or three-dimensional in form, although most studies have used two-dimensional grids. An illustration of a 7 X 7 = 49 cell grid of square cells occupied by two types of ingredients, A and B, is shown in Figure 5.1.

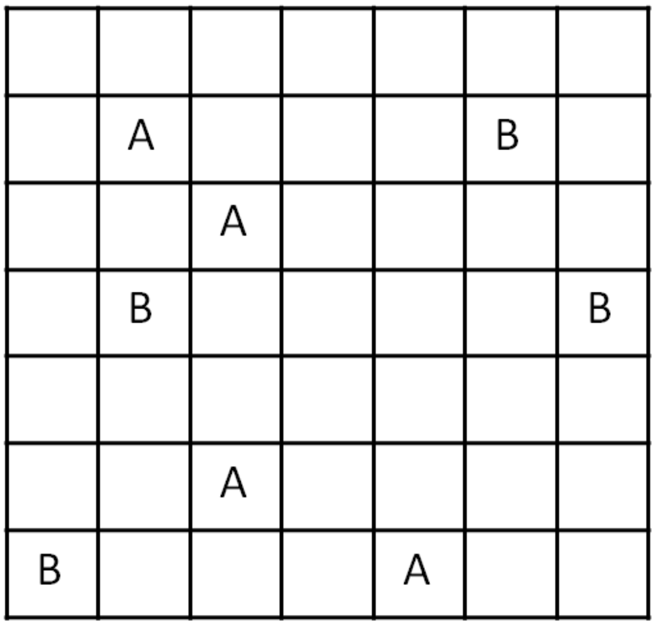


Figure 5.1: A two-dimensional cellular automata grid with two sets of occupied cells of different states, A and B. The unoccupied cells are blank.

The cells themselves can take a variety of shapes. They can be triangles, squares, hexagons, or other shapes on the two-dimensional grid, with square cells being the most common. Each cell in the grid can normally exist in a number of distinct “states” which define the occupancy of the cell. The cell can be empty or contain a specific ingredient, where the ingredient, if present, might represent a particle, a type of molecule or isomer, or a particular molecular electronic state. The interactions of an ingredient with other ingredients take place at the cell edges.

The movement and other actions of an ingredient on the grid are governed by rules, and these rules depend only on the nature of the cells in its neighbourhood. Various types of neighbourhood can be utilized. The most common neighbourhood used in two-dimensional cellular automata is called the von Neumann neighbourhood. For a cell, A, as shown in Figure 5.2(a), the von Neumann neighbourhood refers to the four B cells adjoining its four faces. Another common neighbourhood is the Moore neighbourhood which, as shown in Figure 5.2(b), refers to the eight B cells completely surrounding cell A, including those cells on the diagonals. Another useful neighbourhood is the extended von Neumann neighbourhood as shown in Figure 5.2(c), where the four C cells lying just beyond the four B cells of the von Neumann neighbourhood are included.

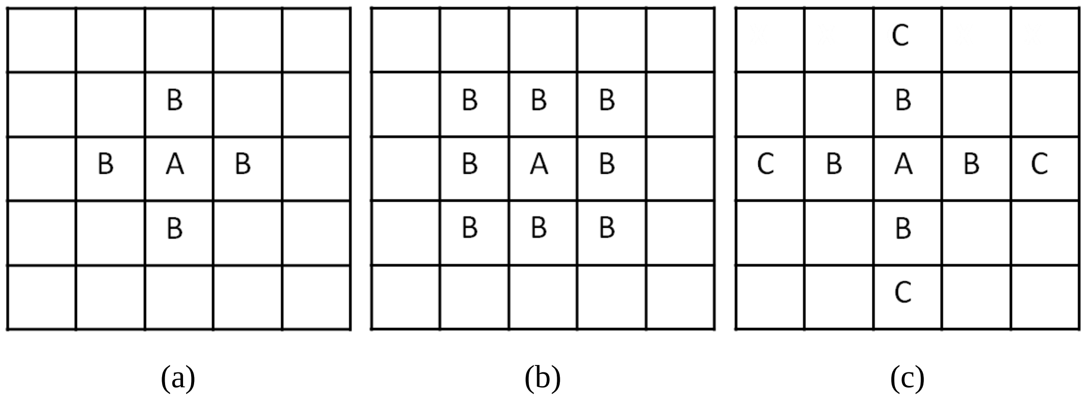


Figure 5.2: Cell neighbourhoods: (a) the von Neumann neighbourhood, (b) the Moore neighbourhood, and (c) the extended von Neumann neighbourhood of cell A.

Different types of rules govern the behaviours of the ingredients on the grid, and thereby the subsequent evolutions of the CA systems. The key features of rules are that they are local, involving only an ingredient itself and possibly the ingredients in its immediate neighbourhood, and that they are uniformly applied throughout the CA simulation.

Having defined both the grid type and size and the governing rules for a simulation, next the remaining conditions of the simulation need to be defined. These include (1) the natures and numbers of the starting ingredients, (2) the configuration of the initial state of the system, (3) how many runs of the simulation are to be carried out, and (4) the length of the run, i.e., how many iterations they should include.

Before the simulation, it is necessary to define the starting condition of the system. Here one first declares what types of ingredients should be present at the start of the run and how many of each type should be present. The ingredients are customarily distinguished on the computer screen by different colours. In order for ingredients to move on the grid, there must be empty cells available to accommodate them.

The default condition for the placement of the starting ingredients is to position them randomly on the grid. For some applications, however, a different distribution might be needed.

The output of a CA simulation carried out on a computer comes in two different forms: a visual output is displayed on the computer screen, and numerical data summaries compiled in output files that are generated during each run. The visual output allows the observer to follow the system as it evolves, and can be very helpful in comprehending the overall process of the system’s evolution. The data summaries in the output files are more suitable for quantitative analysis of the details of this evolution.

5.3.2 Our Bleeding Implementation based on CA

We use two-dimensional grid for our implementation. Experimental and numerical studies show that the von Neumann neighbourhoods can produce satisfactory results and performance in cellular automata surface flow simulations (Mei *et al.*, 2007). Thus we adopt the von Neumann neighbourhood for our implementation.

The effect we want to achieve consists of pigments spreading in every possible direction on a surface driven by the gravity force. So firstly a per-pixel gravity value is calculated according to the normal map of the surface and this gravity map is saved out to a texture for later animation of the blood.

With a normal map consisting of per pixel normals stored in an RGB texture, the direction of the surface points at each point is defined. One of the problems of using normal maps to define gravity is that we need space transformation moving gravity into a tangent space. Once the gravity is defined in a tangent space, the **x** and **y** components of the gravity value serve as a 2D directional vector for gravitational movement.

To determine the final directional vector, the sum of this 2D gravity vector and the **x** and **y** components of the pixel's normal is used. This final 2D vector then yields the direction that the fluid will move across the surface in the tangent space. As shown in Figure 5.3, the **z** value is ignored and the **x** and **y** components are used to dictate direction. This process is applied to each point across the surface to determine a unique gravity vector for each point and the **x** and **y** components are stored to the **r** and **g** channels of the gravity map respectively.

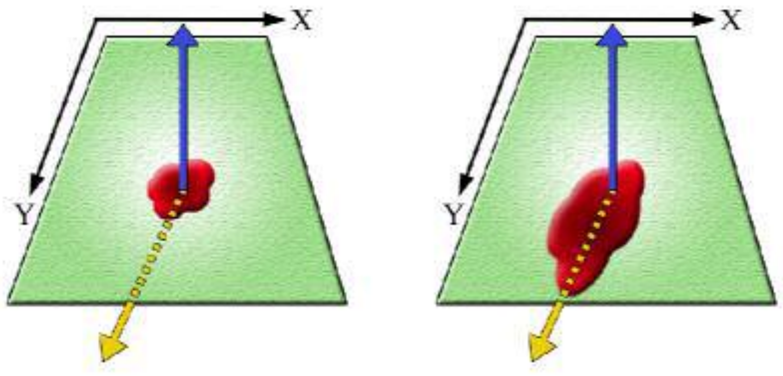


Figure 5.3: The direction of fluid move across the surface.

Following the CA paradigm, the state of each cell is determined by a scalar $w(x,y)$, representing the amount of blood in our case. The amount of blood w is stored in a height map, which is defined in the unused **b** component of the gravity map. In every frame, the height value of the blood at each texel as well as the 2D gravity vector are read out to calculate the new state (the amount of blood w) of the current texel.

New blood appears on the organ due to the surgical cutting, which is defined *blood source* in our implementation. We need to specify the location, the radius and the blood intensity of the source. Currently the location of the blood source is the cutting path. So after the cutting, the cutting path is passed as a parameter to the fluid model for blood simulation. The radius indicates the area that will be affected by the blood source. The blood intensity represents the amount of blood arriving during Δt. By controlling the location of the source, the simulated blood will flow along every possible direction evenly or unevenly according to the surface details and the gravity. By changing the radius and the blood intensity, the area affected by blood propagation could be changed, thus the speed of the blood could be controlled. Let $r_t(x,y)$ be the blood arriving at cell (x,y) per time unit, the amount of blood is update by a simple addition:

$$w_{t+\Delta t}(x,y) = w_t(x,y) + \Delta t * r_t(x,y) * K_r \qquad (5.16)$$

where K_r is a global parameter that controls the overall rate of fluid increment.

The updating of the new state consists of two parts: the blood lost due to the gravity in this frame is subtracted from the current texel; and the blood flown into the current texel from its neighbours are added into the current texel. Let's denote the first part as the inflow flux f_{in} from neighbour cells, and the latter part as the outflow flux f_{out} from the current cell.

The fluid flow between cells can then be calculated. Each cell (x,y) has four neighbours which could exchange fluid with each other. It can be assumed that each (x,y) cell has four virtual pipes to the four neighbours which transport fluid outward from the given cell. The neighbouring cells also have four virtual pipes, transporting fluid to opposite directions as illustrated in Figure 5.4. The fluid outflow flux is update with the amount of blood $w(x,y)$ difference between interconnected cells.

Let's denote $f = (f^L, f^R, f^T, f^B)$ as the outflow flux in a given (x,y) cell, where $f^L$is the outflow flux to the left neighbour at (x-1,y), and similarly f^R, f^T, f^B are the outflow fluxes to right, top, bottom directions, respectively. The change of f^L can then be calculated as:

$$f^L_{t+\Delta t}(x,y) = max(0, f^L_t(x,y) + \Delta t * g * \Delta w^L(x,y) \qquad (5.17)$$

where g is the gravity vector pointing in the direction of the current cell and $\Delta w^L(x,y)$ is the height difference between cell (x,y) and its left neighbour (x-1,y):

$$\Delta w^L_t(x,y) = w_t(x,y) - w_{t-1}(x,y) \qquad (5.18)$$

f^R, f^T, f^B are computed in a similar way.

The blood height is updated with the new outflow flux field by collecting the inflow flux f_{in} from neighbouring cells, and sending the outflow flux f_{out} away from the current cell. For cell (x,y), the new volume change for the blood is:

$$\Delta V(x,y) = \Delta t * \left(\sum f_{in} - \sum f_{out}\right) = \Delta t * (f^R_{t+\Delta t}(x-1,y) + f^T_{t+\Delta t}(x,y-1) + f^L_{t+\Delta t}(x+1,y) + f^B_{t+\Delta t}(x,y+1) - \sum_{i=L,R,T,B} f^i_{t+\Delta t}(x,y) \qquad (5.19)$$

The blood height in each cell is then updated as:

$$w_{(t+\Delta t)}(x,y) = w_t(x,y) + \Delta V(x,y) \qquad (5.20)$$

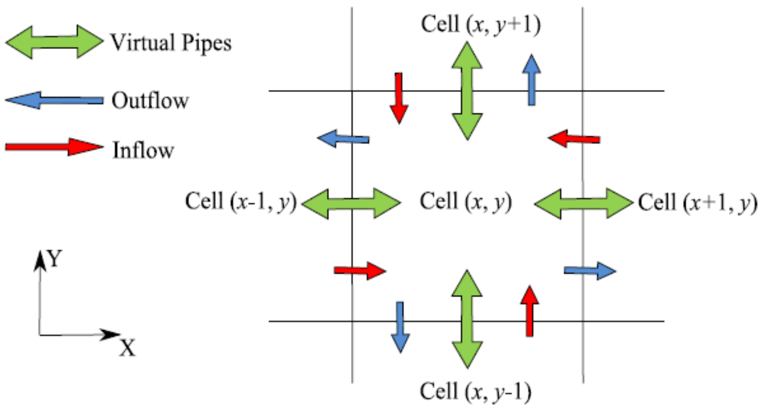
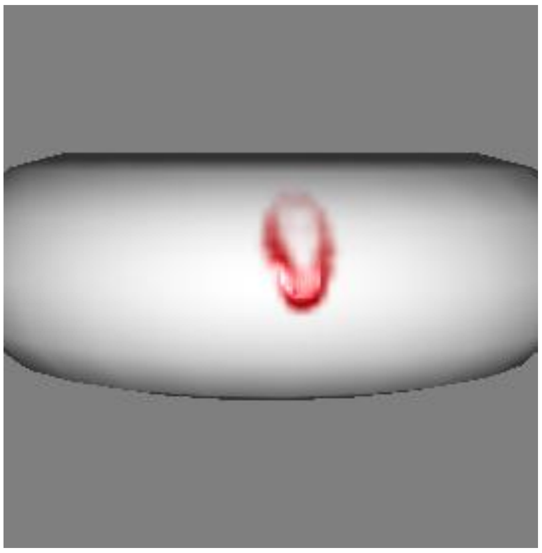
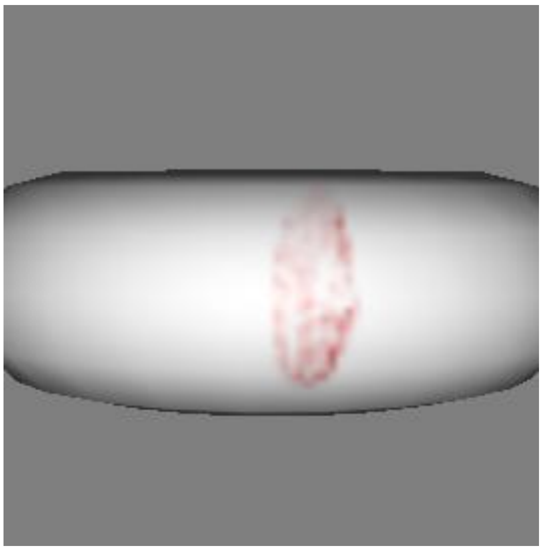


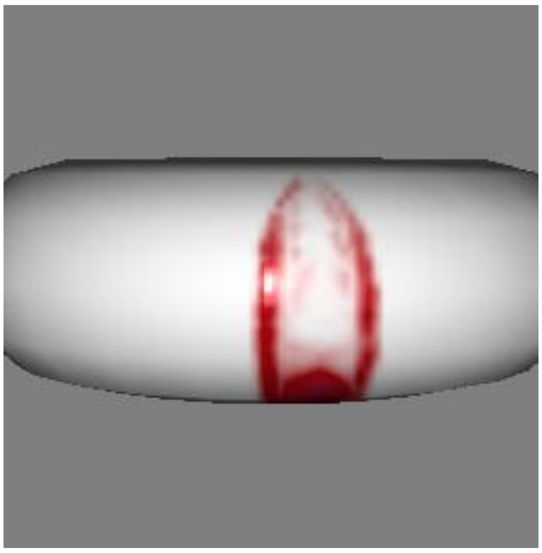
Figure 5.4: Each cell is connected to its four neighbours through virtual pipes and exchange fluid with them.

For any fluid simulation method, boundary conditions should be taken into consideration. Since the flow is simulated on a 2D grid, we assume no fluid can flow out of the grid (“no slip” boundary condition). In the CA model, we specify the outflow flux on boundary cells to satisfy the conditions. For cells (x,0) on the left boundary, the outflow flux to the left neighbour $f^L(x,0)$ should be set to zero. Similar rules apply for the other boundaries.

In order to obtain the effect of blood flowing continuously, at every time step the current texel blood height is compared to a predefined threshold, which represents the minimum height for the blood to exist and flow. If the current value is greater than the threshold, the generation of the gravity map is again performed for the next frame. This cycle continues until there are no longer any texels with blood high enough to be visualized and moved. By the end of this, the blood either stoped flowing leaving a trail on the surface, as shown in Figure 5.5 (a) and (b), or the blood accumulated in the boundary cell, as shown in Figure 5.5 (c) and (d).


(a)


(b)


(c)

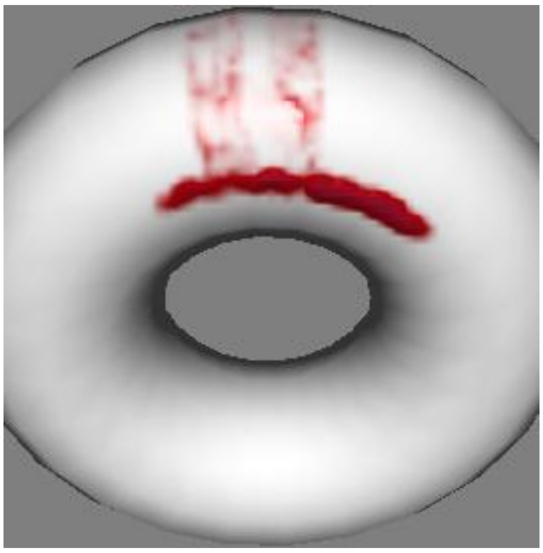

(d)

Figure 5.5: Two cases when blood stops flowing.

The states are updated by a render-to-texture operation using the previous state value and the neighbouring information as input during the simulation and stored in a 2D texture map. The render-to-texture of each time step performs 4 suboperations: Neighbour Sampling, Computation on Neighbours, New State Computation, and State Update. Figure 5.6 illustrates the mapping of the suboperations to graphic hardware. Neighbour sampling and Computation on Neighbours are performed by the programmable texture mapping hardware. New State Computation performs arithmetic on the results of the previous suboperations using programmable texture blending. Finally, State Update feeds the results of one frame to the next by rendering or copying the texture blending results to a texture.

Neighbour Sampling: since the state is stored in textures, neighbour sampling is performed by offsetting texture coordinates toward the neighbours of the texel being updated. For example, to sample the four nearest neighbour nodes of node *(x,y)*, the texture coordinates at the current quadrilateral are offset in the direction of each neighbour by the width of a single texel. Texture coordinate interpolation ensures that as rasterization proceeds, every texel's neighbours will be correctly sampled.

Computation on Neighbours: as described above, in order to compute how much blood will be lost and flown into the current texel, complex functions of the sampled neighbours have to be computed. These functions are computed ahead of time and stored in a texture for use as a lookup table. Our approach relies on dependent texture lookups provided by the programmable texture shader functionality of GPU.

New State Computation: once we have sampled the values of neighbouring texels and used them for function table lookups, we need to compute the new state of the lattice. We use programmable hardware texture blending to perform arithmetic operations. The results of these computations are written to the frame buffer.

State Update: once the new state is computed, we must store it in a state texture. In our current implementation, we copy the newly-rendered frame buffer to a texture using the glCopyTexSubImage2D() instruction in OpenGL. Since all simulation state is stored in textures, our technique avoids large data transfers between the CPU and GPU during simulation and rendering.

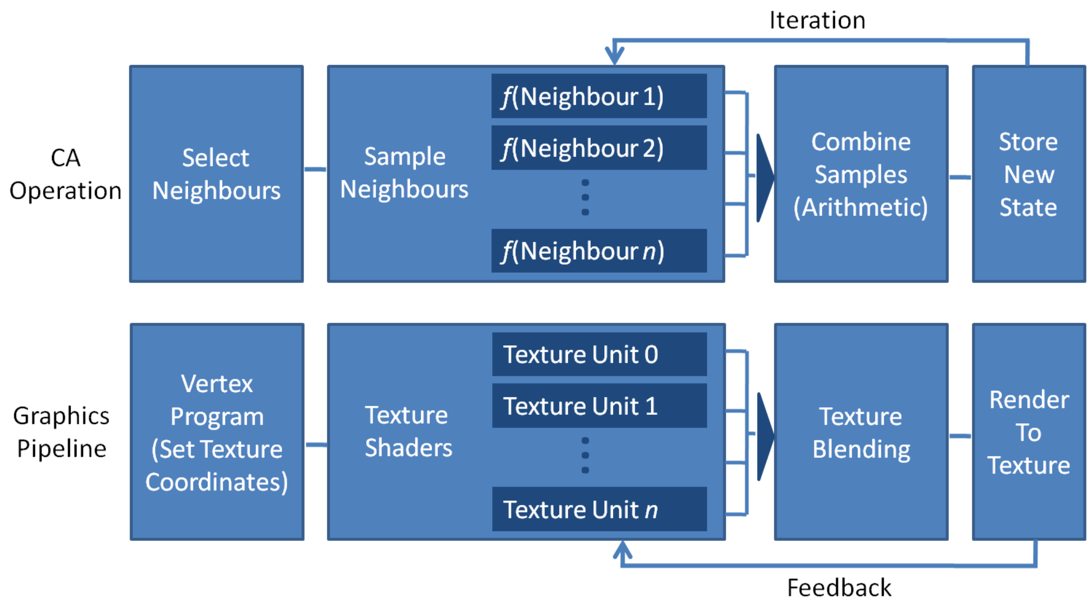


Figure 5.6: Components of a CA operation map to graphics hardware pipeline components.

The final result of the fluid simulation is a continuously changing height map with a 2D gravity-induced directional vector defined at each point. Visually, the goal is to produce the effect of a thick fluid flowing across a surface. The height map contains information regarding the intensity of the colour of each point. This value is multiplied with a given blood colour to yield a final blood colour contribution which is added to the colour of the original surface material. The blood colour is obtained from real surgery videos for high fidelity in realism.

Viscosity has the effect of smoothing the velocity of fluid. It corresponds to the informal notion of "thickness". For example, honey has a higher viscosity than water. To create a more viscous fluid that responds in a realistic manner to surface details, the gravity vector is raised to the third power. It creates a sort of attenuation that ramps the effect of gravity. Figure 5.7 demonstrates this effect. Figure 5.7(a) is the snapshot of blood flowing on the surface and Figure 5.7(b) is the wireframe of the fish. By rotating the wireframe a little bit, as shown in the Figure 5.7(c), we can see that the surface has the sharpest slope around the green area, and accordingly in the Figure 5.7(a) the blood tends to accumulate around the corresponding area. This is because for surface where the slope is sharper, the gravity vector pointing to the below neighbour is less, thus the amount of blood falling into the below texel is less and the blood is more likely to stay in the current texel.

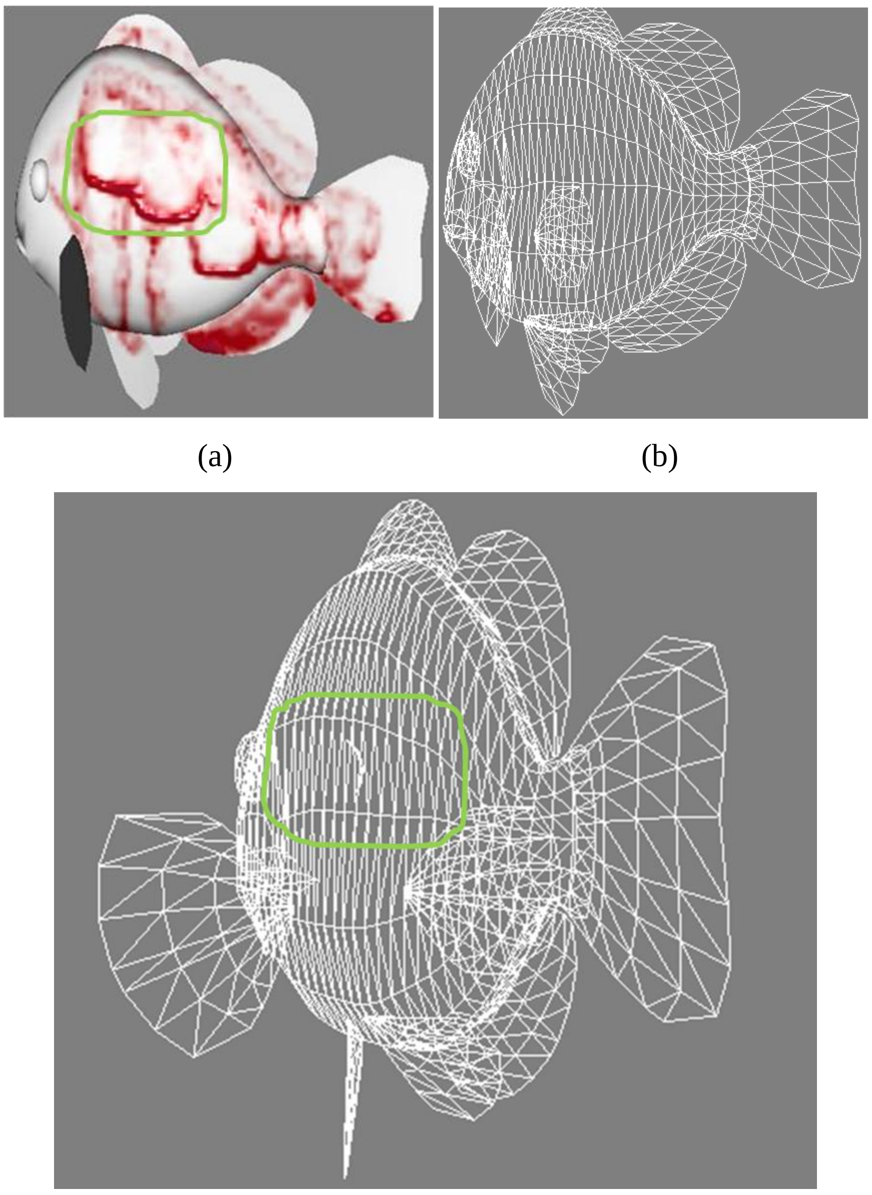


Figure 5.7: An example to demonstrate the viscosity effect of the fluid.

In order to simulate the volume and thickness of the blood, the intensity of the blood colour is increased. To do that, the colour contribution of the surface is decreased by a factor of (1-BloodHeight). For added realism, compute deltas for the blood height of each texel based on its neighbours. Derive *dx* = *LeftHeight* – *RightHeight* and *dy* = *BelowHeight* – *AboveHeight*. These values are then added to the surface normal to give the effect of a thick fluid.

To get the diffuse colour, the blood colour is blended with the surface colour; its transparency varies with the blood height *w*. Various specular coefficients can be used for organ surfaces and blood surface respectively. Phong illumination model is employed for per pixel lighting.

When implementing the CA method, it is easy to notice that storing cells for even a modest-size level consumes a huge amount of memory, and the processing and memory-bandwidth requirements become severe. The trick is to not store or process cells that are not participating in any interesting activities. An octree is ideally suited for such an arrangement, specifically a dynamically-allocated octree. Next section will discuss the implementation of the octree structure for speeding up the fluid simulation further. The GPU is employed as a tree lookup generator in the simulation. The algorithm for the texture mesh is developed based on the octree texture from the fragment program using the tree lookup as suggested in the work by Lefebvre *et al.* (2005). The octree's leaves store the index of a pixel in the form of the three 8-bit values (in RGB Channels) and the alpha channel is used to distinguish between a pointer to a child and a leaf of the octree.

5.3.3 A GPU-Accelerated Hierarchical Structure: The N^3-Tree

5.3.3.1 Definition

An octree is a regular hierarchical data structure. The first node of the tree, the *root*, is a cube. Each node has either eight children or no children. The eight children form a 2x2x2 regular subdivision of the parent node. A node with children is called an *internal node*. A node without children is called *a leaf*.

In an octree, the resolution in each dimension increases by two at each subdivision level. Thus, to reach a resolution of 256x256x256, eight levels are required ($2^8= 256$). Depending on the application, one might prefer to divide each edge by an arbitrary number N rather than 2. Therefore a more generic structure is called an N^3 *-tree*. In an N^3-tree, each node has N^3 children. The octree is an N^3-tree with $N = 2$. A larger value of N reduces the tree depth required to reach a given resolution, but it tends to waste memory because the surface is less closely matched by the tree.

5.3.3.2 Implementation

To implement our CA method as a hierarchical tree on a GPU, how to store the structure in texture memory and how to access the structure from a fragment program need to be defined.

A simple approach to implement an octree on a CPU is to use pointers to link the tree nodes together. Each internal node contains an array of pointers to its children. A child can be another internal node or a leaf. A leaf contains only a data field.

The implementation of an octree on the GPU could follow a similar approach. Pointers simply become indices within a texture. They are encoded as RGB values. The content of the leaves is directly stored as an RGB value within the parent node's array of pointers. The alpha channel could be used to distinguish between a pointer to a child and the content of a leaf. The implementation relies on dependent texture lookups (or *texture indirections*). This requires the hardware to support an arbitrary number of dependent texture lookups, which is the case for GeForce FX and GeForce 6 Series GPUs.

An octree comprised of pointers and nodes is relatively easy to create and works well for a CPU based implementation. Unfortunately, this structure is almost impossible to transfer to GPU. In fact such structure is extremely inefficient to use on GPU. A GPU efficient octree is a condensed octree that has been flattened into an array structure. Essentially, all nodes in the octree that are interior nodes of leaf nodes with data are stored in the depth-first order into an array. The leaf nodes, instead of containing data, contain indices into a common array containing all of the surfels. This structure allows for the minimum amount of data to be used to fully represent the octree. This also allows for one direction octree transversal. This means that to find an intersection it is only necessary to increment across the octree once.

The tree is stored in an 8-bit RGBA 3D texture called the *indirection pool*. Each "pixel" of the indirection pool is called a *cell*.

The indirection pool is subdivided into *indirection grids*. An indirection grid is a cube of *N*x*N*x*N* cells (a 2x2x2 grid for an octree). Each node of the tree is represented by an indirection grid. It corresponds to the array of pointers in the CPU implementation described earlier.

A cell of an indirection grid can be empty or can contain one of the following:

- data, if the corresponding child is a leaf
- the index of an indirection grid, if the corresponding child is another internal node

Figure 5.8 illustrates our tree storage representation. We can see that a quadtree is flattened into an array.

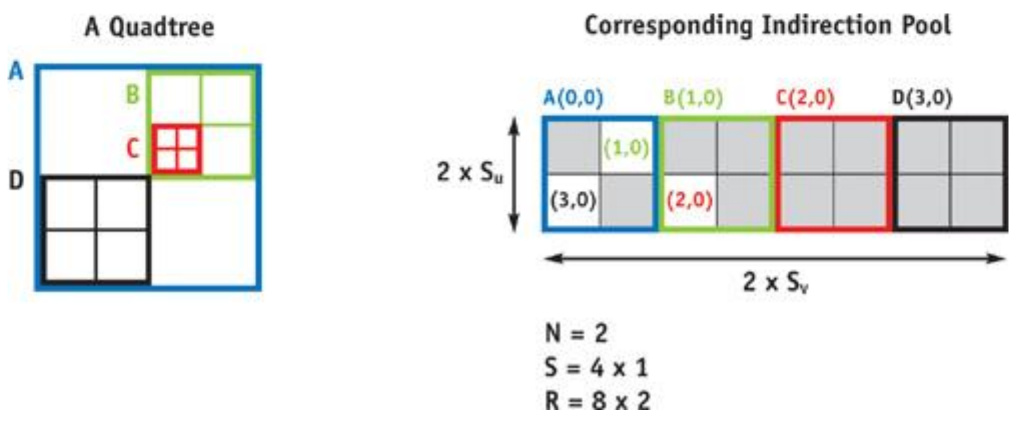


Figure 5.8: Storage in Texture Memory (2D Case).

It can be noted that **S** = S_u x S_v x S_w which is the number of indirection grids stored in the indirection pool and R= (N x S_u) x (N x S_v) x (N x S_w) which is the resolution in cells of the indirection pool.

The data values and indices of children are both stored as RGB triples. The alpha channel is used as a flag to determine the cell content (alpha = 1 indicates data; alpha = 0.5 indicates index; alpha = 0 indicates empty cell). The root of the tree is always stored at (0, 0, 0) within the indirection pool.

Once the tree is stored in texture memory, it needs to be accessed from a fragment program. As with standard 3D textures, the tree defines a texture within the unit cube. We want to retrieve the value stored in the tree at a point $M \in [0, 1]^3$. The tree lookup starts from the root and successively visits the nodes containing the point M until a leaf is reached.

Let I_D be the index of the indirection grid of the node visited at depth D. The tree lookup is initialized with I_0= (0, 0, 0), which corresponds to the tree root. When we are at depth D, we know the index I_D of the current node's indirection grid. How to retrieve I_{D+1} from I_D is explained as follows.

The lookup point M is inside the node visited at depth D. The value stored at the location corresponding to M needs to be read out from the indirection grid ID. To do so, the coordinates of M *within* the node needs to be computed.

At depth *D*, a complete tree produces a regular grid of resolution $N^{D} \text{ x } N^{D} \text{ x } N^{D}$ within the unit cube. This grid is called the *depth-D grid*. Each node of the tree at depth *D* corresponds to a cell of this grid. In particular, *M* is within the cell corresponding to the node visited at depth *D*. The coordinates of *M within* this cell are given by $frac(M \text{ x } N^{D})$. These coordinates are used to read the value from the indirection grid *ID*. The lookup coordinates within the indirection pool are thus computed as:

$$P = \frac{I_D + frac(M \times N^{D})}{S} \tag{5.21}$$

The RGBA value stored at *P* in the indirection pool is then retrieved. Depending on the alpha value, either we will return the RGB colour if the child is a leaf, or the RGB values will be interpreted as the index of the child's indirection grid (I_{D+1}) and continue to the next tree depth. Figure 5.9 summarizes this entire process for the 2D case (quadtree).

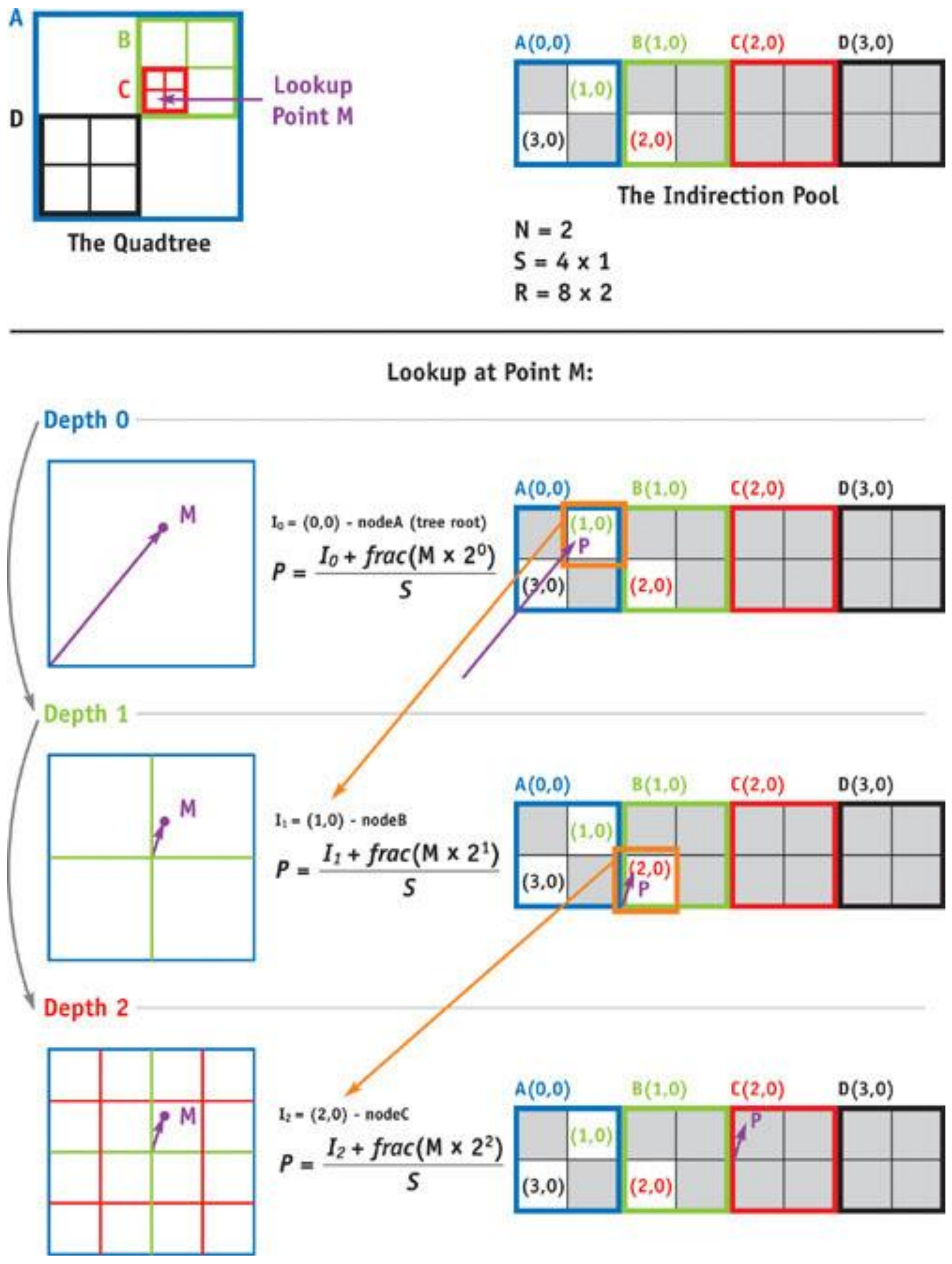


Figure 5.9: Example of a Tree Lookup (Lefebvre *et al.* 2005)

The lookup ends when a leaf is reached. In practice, the fragment program also stops after a fixed number of texture lookups. With most hardware, it is only possible to implement loop statements with a fixed number of iterations. The application is limiting the tree depth with respect to the maximum number of texture lookups done within the fragment program. The complete tree lookup code is shown in Listing 5.1.

```
float4 tree_lookup(uniform sampler3D IndirPool, // Indirection Pool↵
  uniform float3 invS,↵
  uniform float N,↵
  float3 M) // Lookup coordinates↵
{↵
  float4 I = float4(0.0, 0.0, 0.0, 0.0);↵
  float3 MND = M;↵
  for (float i=0; i<HRDWTREE_MAX_DEPTH; i++) {↵
    float3 P;↵
    // compute lookup coords within current node↵
    P = (MND + floor (0.5 + I.xyz * 255.0)) * invS;↵
    // access indirection pool↵
    if (I.w < 0.9)                   // already in a leaf?↵
      I = (float4) tex3D (IndirPool, P); // no, continue to next
depth↵
 #ifdef DYN_BRANCHING // early exit if hardware supports dynamic
branching↵
    if (I.w > 0.9)     // a leaf has been reached↵
      break;↵
#endif↵
    if (I.w < 0.1) // empty cell↵
      discard;↵
    // compute pos within next depth grid↵
    MND = MND * N;↵
  }↵
  return (I);↵
}↵
```

Listing 5.1: The tree look-up Cg code.

The octree representation of the associate geometry is shown in the Figure 5.10. The blood source is passed as a parameter to the fluid model to calculate the intersections with the surface, as shown in Figure 5.10(b). These intersections might contain maximum 27 (3*3*3) cubes. However, the octree would determine the actual number of cubes according to the shape of the surface, as the texture information is only stored around the surface. As shown in Figure 5.10(c), only 9 green cubes are actually identified for the blood propagation.

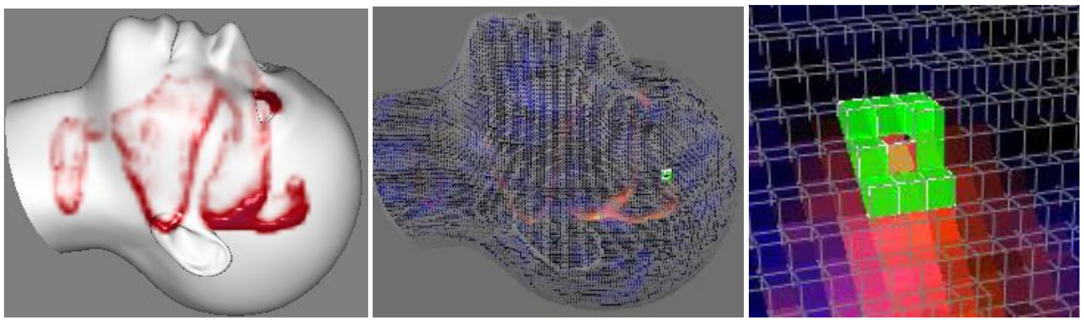

(a) (b) (c)

Figure 5.10: The octree mapping of the releasing fluid; (a) fluid moving over the geometry surface; (b) the octree structure of the mesh (the green cube indicates the current source of the fluid); (c) a close-up of the octree.

5.3.4 Experimental Results

All the results discussed in this thesis are implanted on a 2.4 GHz dual-CPU personal computer with 2GB memory and a NVIDIA Quadro FX 4600 graphics card. SensAble Technologies' PHANToM premium 1.5 is used to provide force reflectance and feedback for haptic interaction. The PAHNToM provides 6 degrees of force feedback allowing the user to explore object models using the sense of touch. The force feedback also provides valuable sensory feedback to the surgeon during simulation of tissue deformation and cutting. The visual and haptic rendering were developed based on the open source H3DAPI and Volume Haptics Toolkit (VHTK), both are created by SenseGraphics AB.

By utilizing the GPU and octree structure, our non-physics fluid model is able to simulate fluid flowing along a surface very realistically without adding computation load to the CPU. Figure 5.11 shows some snapshot of the fluid simulation by adding the fluid on the surface meshes interactively.

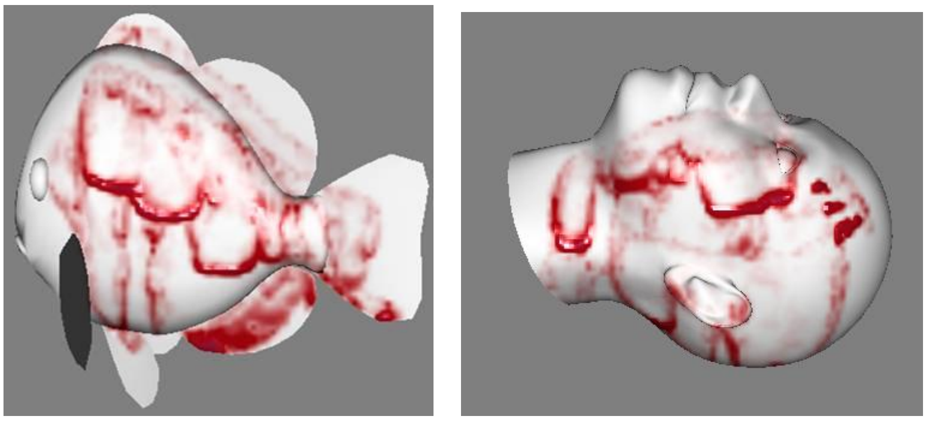


Figure 5.11: fluid on different mesh models

Figure 5.12 is an example to show that our fluid model is totally following the topology of the surface, not only capable of flowing along a smooth surface, but also a complex surface such as concave surface.

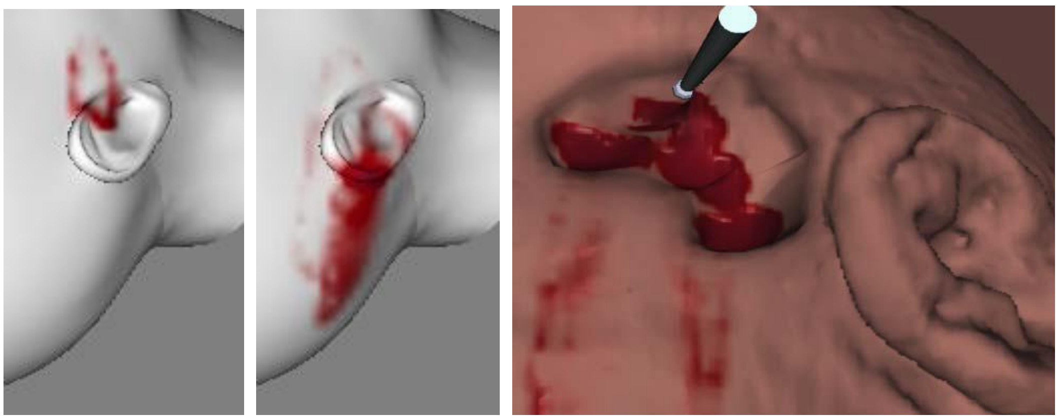
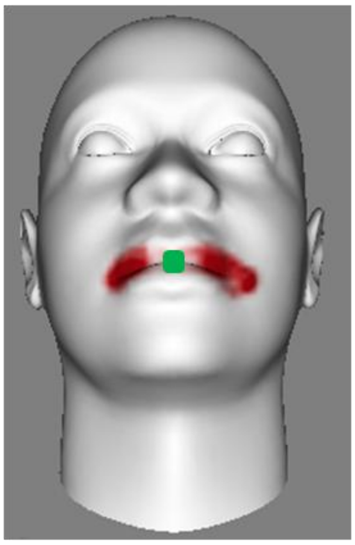
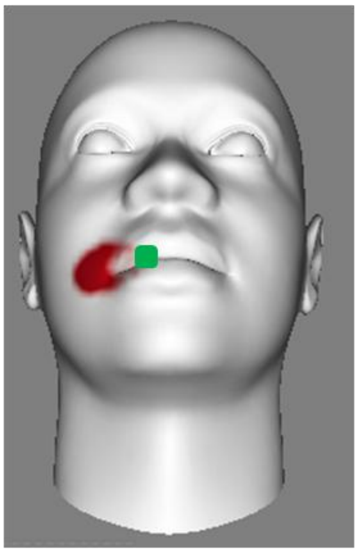


Figure 5.12: Fluid flowing on complex surfaces

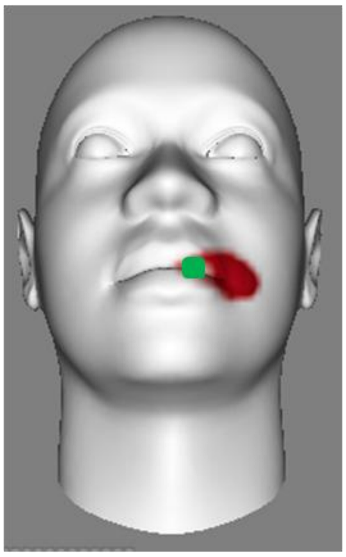
Figure 5.13 demonstrates that our fluid model is able to simulate the fluid phenomena in real life. The green cube in Figure 5.13 represents the blood source. As it can be seen from Figure 5.13(a), when the source is in the top most position on the mouth, the fluid will flow along all possible directions evenly following the gravity force. When the source is put in a lower position as in Figures 5.13(b) and (d), the fluid will flow totally downwards to the left or right respectively. Figures 5.13(c) and (e) show that when the source is put in the second most top position, the fluid will flow unevenly towards the left and right direction according to the position of the source.



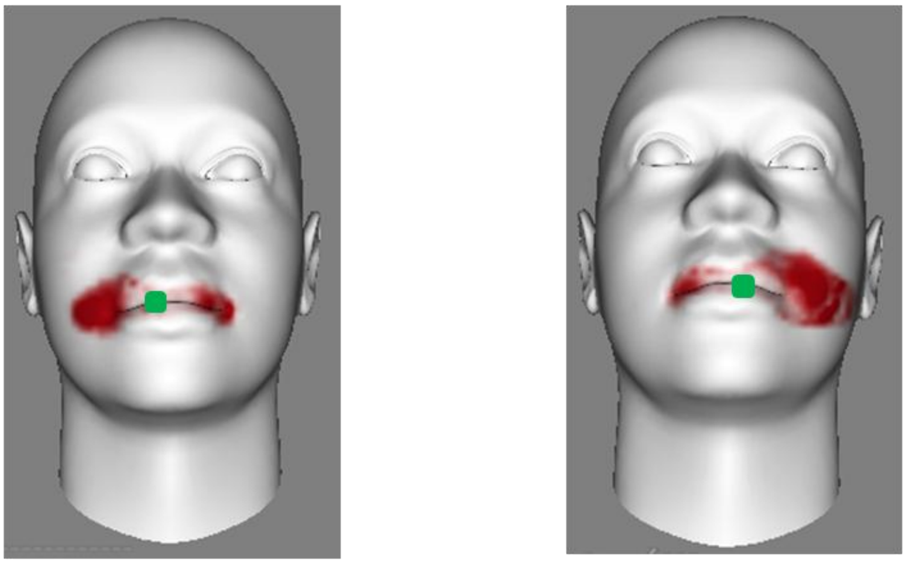
(a)



(b)



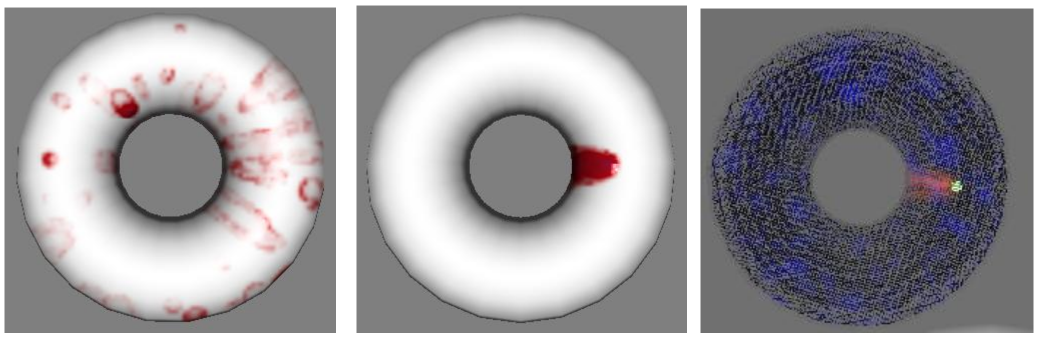
(c)



(d) (e)

Figure 5.13: Our fluid model resembles real life fluid phenomena as much as possible.

Figure 5.14 exhibits the scalability of our fluid simulation by using different types of fluid source. Figure 5.14(a) shows the top view of random blood drops as the fluid source. Each blood drop falls on the surface with a random distribution. Figure 5.14(b) shows the top view of fixed fluid source where the location of the source can be randomly indicated by the user, in this case shown as the green cube in Figure 5.14(c). The volume of fluid and the propagation speed can also be easily controlled by specifying the radius and the intensity of the source.



(a) (b) (c)

Figure 5.14: Different fluid sources.

Figure 5.15 shows a few snapshots of surgical cutting on a surface mesh followed by blood flowing over the surface. The surface is an iso-surface extracted from the corresponding volumetric data. The texture coordinates are defined by indexed triangle mesh with customizable colour scheme. The colour scheme is designed to be in accordance with the corresponding volumetric data.

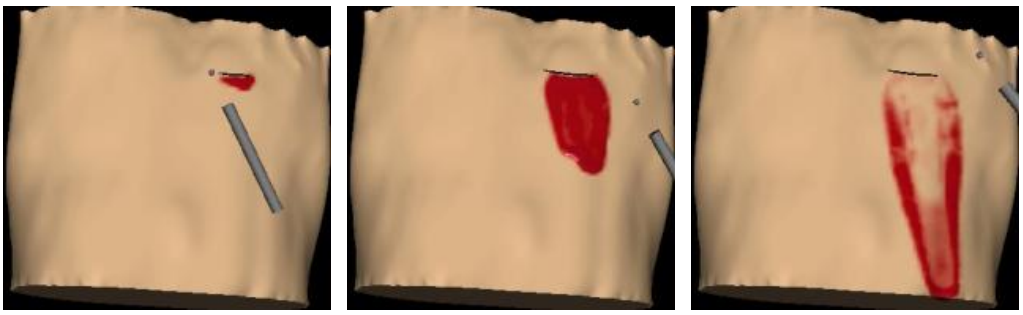
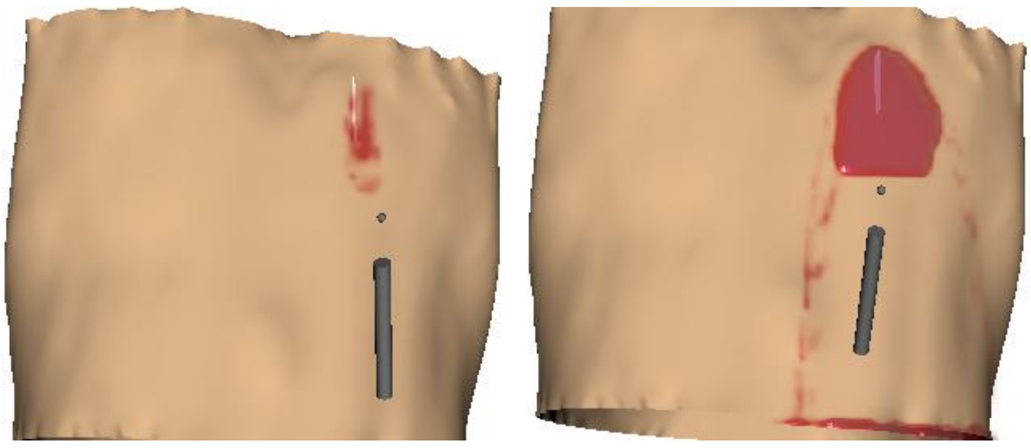


Figure 5.15: Blood flowing on a torso surface.

According to the types of cuts, different kinds of bleeding could be simulated by changing the amount of blood and the radius of the blood source. For example, if the organ is cut with a very light force, a very small amount of fluid with a low flowing speed will be produced with a thin cut. If the object is cut with a large force, sometimes a huge amount of blood will spout out and spread the cutting area immediately. Figure 5.16 shows such two examples with different cut widths of 0.003 and 0.3 respectively.



(a) (b)

Figure 5.16: Different blood amount controlled by cut width. (a) With a cut width of 0.003. (b) With a cut width of 0.3.

By incorporating the volumetric model underneath, the hybrid model is able to simulate realistic surgery cutting followed by blood flowing in the operation area including both the surface and the volume. The surgery simulation starts with a progressive thin cut by haptic interaction on the outer surface and the inner volume simultaneously, and then bleeding occurs along the cutting path and the blood spreads around the cut area rapidly. This is achieved by pass the cutting path as a parameter to the fluid model as the source of the fluid and the blood will propagate along every possible direction driven by the gravity. The operation area is often cleaned by sucking or wiping the blood away within a short period of time. We implement this effect by clicking a button to clean all the previous blood tracks. The haptic device is then used as a retractor to open up the cut resulting in the groove generation and the display of the interior structures. Fluid simulation is triggered again by the pulling process. During the pulling, the surface is deformed interactively according to the shape of the inner groove. By passing the position of the groove as the location of the blood source to the fluid model, the bleeding could happen in the groove and the blood keeps flowing from the inner wound onto the outer surface following the changing topology of the surface. In this way, it looks like the blood is blocking the vision of the whole wound area which is just one of the purposes of simulating fluid in virtual surgery. A spilling fluid from the cut is assumed as a droplet liquid that travel over the mesh. When the dropped fluid flows on a surface, some amount of fluid remains behind because of surface absorption and wetting. The remaining fluid will merges into the other fluid that spills from the wound subsequently. Figure 5.17 shows some snapshots of cut on the torso with fluid flowing around the cutting area.

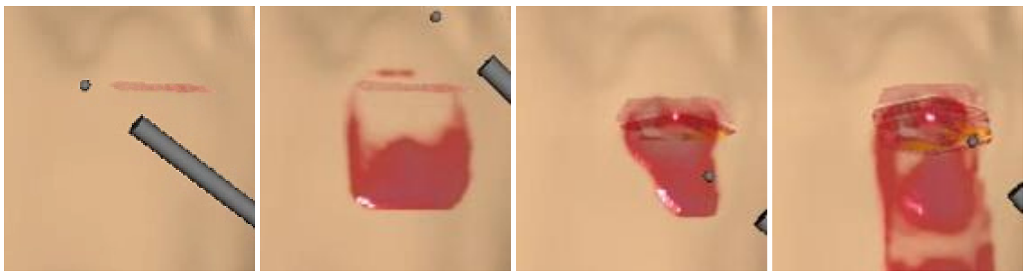


Figure 5.17: Cutting and bleeding for cut on the torso.

Another surgical scenario being simulated is the bone drilling operation. The skin is opened up to reveal the underneath bones, which will be drilled by the haptic device, as depicted in Figure 5.18. The drilling process is also accompanied with bleeding effect.

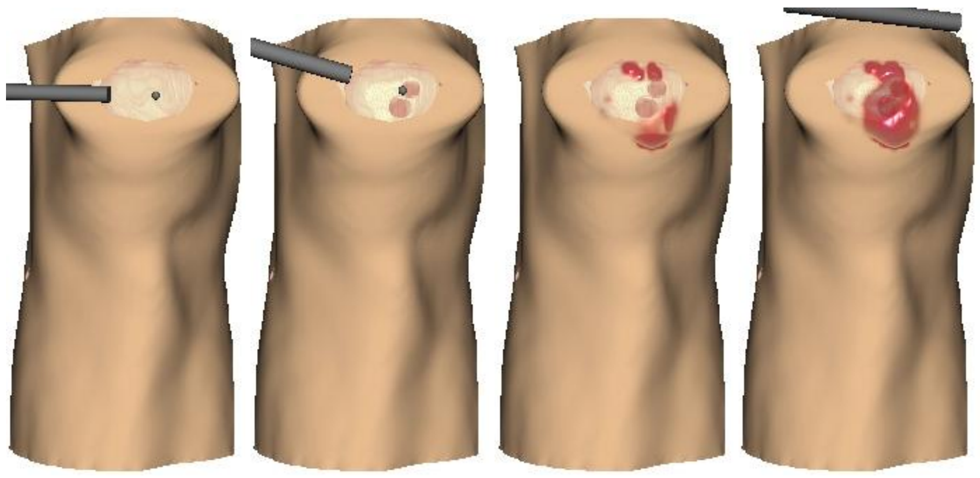
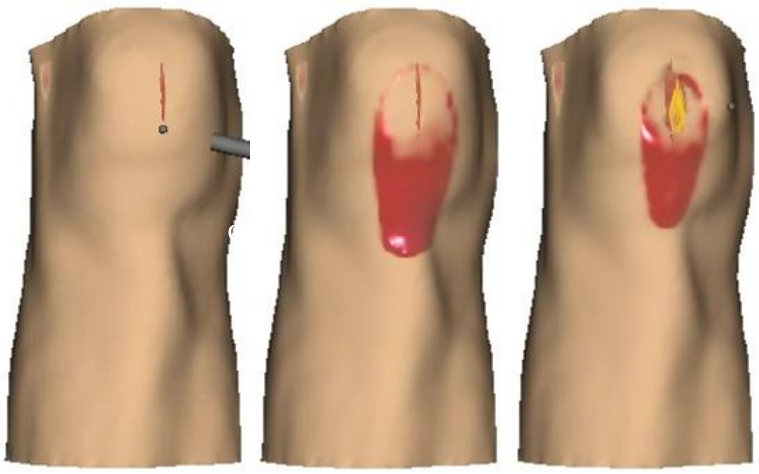


Figure 5.18: Drilling process with bleeding effect.

Our simulation system is also applied to simulate the cutting for a knee surgery. The whole process is shown in Figure 5.19, in which the haptic device is modelled as the surgical instrument and force feedback are sent to the user during the cutting and the widening. The knee is placed vertically so that the blood is flowing down along the surface because of gravity.



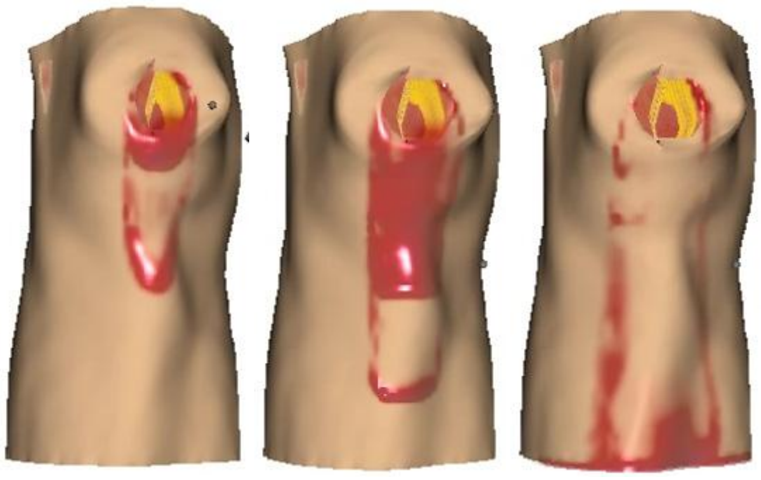


Figure 5.19: Cutting and bleeding for knee surgery

Figure 5.20 also shows another example when the knee is tilted with certain angle. In this case, the blood flows in every possible direction unevenly.

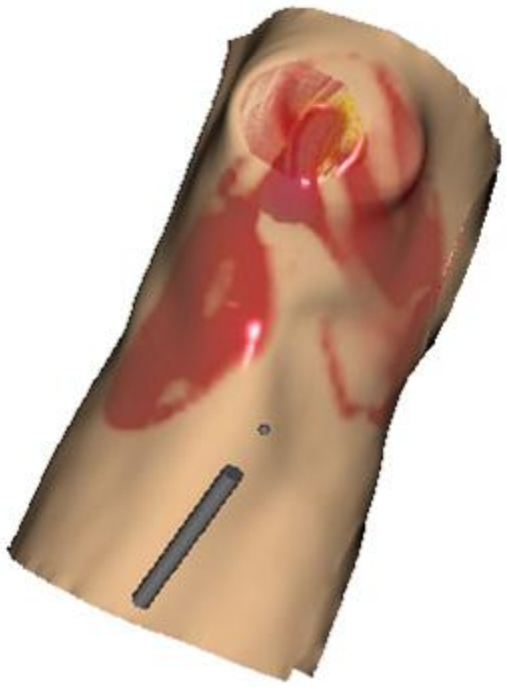


Figure 5.20: Cutting and bleeding on a tilted knee.

5.3.5 Discussion on Performances

The overall system simulation includes several surgical scenarios such as knee surgery and cutting on the chest. In order to observe the influence of fluid model to the performance of overall system, we break the experiments into three groups and test the performances when there is only cutting simulated or only blood simulated or both of them are simulated. The average performances observed during the experiments are provided in the Table 5.1. The data is observed based on the frame rate on G_{fps} (Graphic frame rate per second) and H_{fps} (Haptic frame rate per second). Different mesh sizes are examined. (c) is the frame rate for cutting without bleeding, and (f) is the frame rate for blood simulation without cutting, and (cf) means the frame rate with both cutting and bleeding. As shown in the table, the graphic frame rate remains stable for various mesh sizes either with fluid dynamic included or not. However, when a mesh cutting through haptic is included, the simulation frame rate fall into 12 fps for the triangles greater than 30000, no matter whether the fluid dynamic is included or not. We can see that the fluid model does not impact on the interaction speed but the cutting process does. This is because our fluid model is computed and rendered totally on GPU, and the re-meshing processes and mesh deformations need more computation on the CPU. Same trend can be observed from the frame rates on haptic rendering, although they remain to be above 950 fps which are acceptable. The low rendering frame rate for large data with both cutting and fluid included could be improved by concentrating only on affected area in the future. At this time, we only focus on the influence of fluid on the system performance.

Mesh	No. of Triangles	G_{fps}	$G_{fps(f)}$	$G_{fps(c)}$	$G_{fps(cf)}$	$H_{fps(c)}$	$H_{fps(f)}$	$H_{fps(cf)}$
Chest	37000	59	58	13	12	953	998	950
Knee	16742	60	59	28	25	980	1000	980
Head	12280	60	60	28	28	989	1001	983
Fish	2128	60	60	59	58	1001	1001	1000

Table 5.1: Mesh sizes versus frame rate per second (fps).

5.4 Chapter Summary

The need to incorporate the effects of bleeding in surgical simulations has been well recognized and several previously developed simulators include such effects. However, most of these methods are not suitable for real-time environments, as they are intended for animations in which each frame can be pre-computed for many hours using high-performance computers before putting together to form a smooth animation. We develop a non-physical simulation for the effect of blood flowing along the anatomical surface. It utilizes the GPU parallel architecture for both the computations and rendering of fluid so that the CPU could focus on other time-critical tasks. The fluid model is a texture based approach that uses pixel and vertex shader level techniques in the GPU. The actual simulation is based on cellular automaton which is a discrete model for the description of physical dynamic systems regulated only by local laws. A gravity-vector and a normal map of the surface are used to calculate the amount of blood for every pixel. Finally the blood is blended with the actual surface and they are rendered to the screen. To save up space and improve efficiency, we use octree as our structure for texture mesh, since octrees are one of the most efficient spatial data structures that can be implemented. Some snapshots and performance discussions of our surgical simulator have been provided in Section 5.3.4 and 5.3.5, which demonstrate that our fluid model could produce a realistic bleeding simulation for virtual surgery while keep the whole system running in real-time. Such a system can be incorporated into a comprehensive surgical simulation system for the enhanced realism of virtual surgery.

Chapter 6 Conclusion

This thesis has proposed a VR-based surgical simulation system for simulating surgery cutting and bleeding effect based on some novel algorithms.

In Chapter 3, novel algorithms for surgical cutting simulations are proposed based on a hybrid deformable models consisting of surface and volumetric data. 3D node snapping and topology modification approaches are presented to generate the smooth cutting on the outer surface. The cut can be manipulated and widened up to reveal the heterogeneous interior structures and materials of the underneath volumetric deformable models. The volumetric model can also be deformed interactively by the haptic device to generate the cutting gutter by a modified ChainMail algorithm. These two data models are integrated seamlessly together so that the cutting and deformation on these two models can progressively synchronize with each other. Drilling effects are also implemented by removing the contacting voxels on the underneath volumetric model to present the extendibility of our model.

In Chapter 4, haptic interactions during the surgery cutting and bone drilling are discussed. Both the surface haptics and haptics on volume data are explored. Haptic device is modelled as different surgical tools for surface cutting, surface manipulation, and volume deformation. Force feedbacks are calculated in every process accordingly to provide high fidelity for the users so that they can conduct the virtual surgery process as realistic as possible.

To greatly enhance the realism of the surgical simulation, a fluid model proposed in Chapter 5 is integrated with the surgery cutting for simulating the bleeding effects in real time. The fluid model uses a shader to simulate the motion of a fluid over a dynamic surface. The actual simulation is done by a cellular automaton residing on the changing surface of an object and is done in a pixel shader. It is a self-producing process. By passing a gravity-vector to the pixel shader and a normal map that defines the details of the surface, for every pixel the shader looks at its surroundings to see whether there is any blood in the vicinity. If there is, the orientation of the surface and the gravity are used to calculate whether the blood would reach the current pixel. In each frame, the final step is to combine the blood texture, which is dynamically updated, onto the actual object and render this to the screen. By taking advantage of the GPU parallelism, the fluid simulation can be created and run in real time without the need of involving a supercomputer. GPU based octree texture data structure has also been adapted to reduce the memory usage and speed up the whole system. The blood will flow on the anatomical surface from the cutting path when the surgical knife cut the surface to a certain thickness. Unlike previous techniques (Halic et al., 2010; Rianto and Li, 2010) where the fluid only occurs on the static surface, our fluid can starts from the outer surface cutting area, propagate into the inner volumetric model during the process of groove generation, and flow out of the groove to other part of the surface so that the whole surgical scene looks more realistic.

The inner volumetric model is from real patient CT-image data with a large amount of information involving biomechanics human anatomy. Iso-surfaces including foot, knee, and torso have successfully been created from the CT data using the Marching Cubes Algorithm and used as polygonal geometries for the outer surface model. Experimental results demonstrate the effectiveness of our approaches including realistic hybrid surgical cutting simulations and dynamic fluid (blood) visualization. It can be included in comprehensive virtual surgical simulators to provide a realistic, safe, and controllable environment for novice doctors to practice surgical operations at a low cost.

6.1 Summary of Contributions

The contributions of this thesis include the following:

- A hybrid anatomical deformable model is proposed for progressive surgical cutting simulation in a VR based surgical simulators. We try to propose solutions to the three key requirements for simulating surgical cutting:
    - ➢ Realistic. The purpose of simulating surgical cutting is to provide a virtual environment for users to learn/practice procedures and understand operative anatomy in a no-risk environment. Therefore, the simulation should be realistic enough to ensure that the users are exposed to the full range of intraoperative environments. Realistic visual representations have been achieved by surface polygons in our hybrid model, which are overlaid with a 2D mesh to represent the surface appearance of the tissue

by surface rendering. To add realism, the polygons are extracted from the actual patient scan volume data. Various pixel shading and antaliasing techniques can be used to make the surface appear more photorealistic by recreating natural textures and lighting effects and smoothing edges and wet surfaces. Direct volume rendering is employed in our system which could offer a great deal of information about the interior structures to further enhance the realism of the virtual environments. To achieve realistic and immersive virtual environments, however, it is necessary not only to create visually realistic virtual environments but also to create haptic interactions. In our simulation, a user is able to interact with the virtual anatomical model via a haptic force- feedback device. On one hand, this device is used by the user to cut and pull the surface and the volume model. On the other hand, the haptic device emits forces calculated by the simulation to make the user actually feel the forces occurring during the cutting procedure.

- Real-time. In order to achieve real-time performance, the visual refresh rate of the simulator needs to be around 25 Hz so as to give a visually acceptable effect, while the haptic rendering rate needs to be around 1000Hz to give an acceptable feel. The recorded performance reveals that our system has the average thread execution in the range of 20 frames per second for visualization and about 950 frames per second for the haptic force-feedback computation. This proves that the system has real time synchronisation and can manage the advance calculation, in a joint computation of the objects deformations, geometry and voxel manipulations during cutting and drilling and the fluid simulation simultaneously.
- Progressive. Our surface cutting follows the free form path of the user's motion, and generates a minimal set of new elements to replace intersected triangles. Intersected elements are progressively cut to minimize the lag between the user's motion and model modification.

- A light fluid model has been implemented for bleeding simulation. The fluid model is a texture based approach that uses pixel and vertex shader level techniques in the GPU which could create a realistic visual effect without introducing major additional computational cost to the CPU. Although the fluid model is non-physical, it is able to simulate the realistic fluid properties including the viscosity and velocity. Our fluid model does not depend on the size and the geometry of the organ or its associated textures, and can be applied in any complex simulation scene where fluid flowing along surface is needed without significant modification and major reduction in performance.

- A prototype of surgical simulator which integrates the progressive hybrid cutting and fluid simulation has been developed. The prototype is established on a PC to resemble the real-world surgical operation for cutting and drilling. The system is implemented on the open source H3DAPI and VHTK and could provide high-fidelity graphic display and realistic haptic feedback during the surgical cutting and organ deformations interactively.

6.2 Future Work

As previously discussed, our prototype system presented in this thesis is effective and efficient to be incorporated into a comprehensive surgical simulation system. Despite this there is a list of items that could be explored to either improve the current system or be investigated as interesting further study. Foremost is the fact that the proposed system did not implement the interaction between fluid and haptic device. Additionally, multi-fluid interactions are currently not achieved in the proposed system. Furthermore, only one haptic is used in the proposed system. Finally, the efficiency of the proposed approach would ideally be improved.

6.2.1 Fluid-Haptic Interaction

The exploration of force feedback design and computations for haptic instruments has been around for several years. However, only a few are on force feedback design for fluid media. A force feedback integration with interactive fluid model has been introduced by Baxter and Lin (2004). This method enables force and torque generations in virtual painting applications. There is very few other study exploring this field except in Lundin *et al.* (2005) for presenting fluid dynamic data and in (Bhasin *et al.*, 2005) for simulating droplet fluid. Both of them focused more on the fluid visualization rather than its interaction on haptic device. Later, Rianto and Li (2010) implemented a GPU based fluid dynamic simulation system for virtual heart beating surgery which could achieve the interactions between haptic tip and the fluid.

In that system, the haptic tip performs as disturbances over the surface and the fluid react according to the movement of the tip. The involvement of force feedback interactions in fluid visualizations would definitely improve the realism of the virtual environments.

6.2.2 Multi-fluid Interaction

The proposed system does not include multi-fluid interactions at the moment. Multi-fluid interaction is common in a real surgery, such as when there is too much blood blocking the view, the surgeon will douche around the wound and the blood and the irrigation fluid will mix with each other.

6.2.3 Two-handed Haptic Interface

Most of the tasks that we perform in our daily life involve the use of both hands for a wide variety of purposes ranging from a simple pickup task to a more complex and fine manipulation such as surgery tasks. Two-handed manipulation with the haptic interface can control a virtual object more intuitively and efficiently. By grasping a special grip provided by each device, users can interact with virtual objects using both hands and accomplish life-like bimanual tasks in an intuitive manner. In our system, adding a second haptic device can assist the manipulation of the object, either by “holding” the surface mesh during cutting or by affecting the manipulation directly.

6.2.4 Efficiency

The efficiency of the whole system could be further improved by moving the volume rendering into the GPU, which is currently a texture based rendering run on the CPU. Texture based volume rendering could easily exploit the flexible programming model and 3D texturing capabilities of modern graphics hardware as it fits well with the texture support and blending functionalities of GPU.

Apart from those above parts, the fluid model in the proposed system considers haemorrhages as local phenomena and the blood circulation in the whole body is not taken into consideration at the moment. However, vascular anatomy often plays an important role in surgical planning as well as in the execution of the surgery. For example, finding and cauterizing the blood vessels feeding a tumor before its removal will generally reduce blood loss for the patient. Important arteries can also be located near the surgical field and additional care must then be taken to avoid major haemorrhages which could have debilitating consequences for the patient. In other words, simulating blood circulation in real time in a vascular model is extremely beneficial for learning and practicing surgical hemostasis. In the future, we can improve the definition of blood source’s location by incorporating blood circulation, blood pressure and cutting depth, so that the blood will not always appears at the start point of cutting path, but in a more realistic way.

Last but not least, when the system is more mature, evaluation and feedback of the simulation system from real surgeons could certainly help to identify issues of interest and improve its effectiveness.

REFERENCE

Abdulmotaleb E.S. 2012. Haptics rendering and applications. Publisher: InTech.

Ackerman, M.J. 1998. The visible human project. *Proceedings of the IEEE,* 86(3): 504-511.

Adams, B., Keiser, R., Pauly, M., Guibas, L., Gross, M., and Dutré, P. 2005. Efficient Raytracing of Deforming Point‐Sampled Surfaces. In *Computer Graphics Forum*, 24(3): 677-684. Blackwell Publishing, Inc.

Aggarwal, R., Moorthy, K., and Darzi, A. 2004. Laparoscopic skills training and assessment. *British Journal of Surgery*, 91(12): 1549-1558.

Aggarwal, R., Grantcharov, T.P., Eriksen, J.R., Blirup, D., Kristiansen, V.B., Funch-Jensen, P., and Darzi, A. 2006. An evidence-based virtual reality training program for novice laparoscopic surgeons. *Annals of surgery*, 244(2): 310.

Aggarwal, R., Crochet, P., Dias, A., Misra, A., Ziprin, P., and Darzi, A. 2009. Development of a virtual reality training curriculum for laparoscopic cholecystectomy. *British Journal of Surgery*, 96(9): 1086-1093.

Anderson, J.D. 1995. Computational fluid dynamics, 206. New York: McGraw-Hill.

Andersson, L. 2005. Real-time fluid dynamics for virtual surgery. *Master's thesis, Engineering Physics Program, Chalmers University of Technology*.

Bajka, M., Tuchschmid, S., Fink, D., Székely, G., and Harders, M. 2010. Establishing construct validity of a virtual-reality training simulator for hysteroscopy via a multimetric scoring system. *Surgical endoscopy*, 24(1): 79-88.

Basdogan, C., De, S., Kim, J., Muniyandi, M., Kim, H., and Srinivasan, M.A. 2004. Haptics in minimally invasive surgical simulation and training. *Computer Graphics and Applications, IEEE*, 24(2): 56-64.

Batty, C., Xenos, S., and Houston, B. 2010. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In Computer Graphics Forum, 29(2): 695-704.

Baxter, W., and Lin, M.C. 2004. Haptic interaction with fluid media. *In Proceedings of graphics interface 2004*, 81-88. Canadian Human-Computer Communications Society.

Becker, M., and Teschner, M. 2007. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 209-217.

Becker, M., Ihmsen, M., and Teschner, M. 2009. Corotated SPH for deformable solids. In *Proceedings of the Fifth Eurographics conference on Natural Phenomena*, 27-34. Eurographics Association.

Bemelman, W.A. 2009. Mastery of Endoscopic and Laparoscopic Surgery.

Bhasin, Y., Liu, A., and Bowyer, M. 2005. Simulating surgical incisions without polygon subdivision. *Studies in Health Technology and Informatics*, 111: 43-49.

Bielser, D., and Gross, M.H. 2000. Interactive simulation of surgical cuts. In *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference*, 116-442. IEEE.

Bielser, D., and Gross, M.H. 2002. Open surgery simulation. *Studies in Health Technology and Informatics*: 57-63.

Bielser, D., Glardon, P., Teschner, M., and Gross, M. 2004. A state machine for real-time cutting of tetrahedral meshes. *Graphical Models*, 66(6): 398-417.

Brett, P.N., Parker, T.J., Harrison, A.J., Thomas, T.A., and Carr, A. 1997. Simulation of resistance forces acting on surgical needles. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine* 211(4): 335-347.

Bridson, R., and Müller-Fischer, M. 2007. Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page, 1-81. ACM.

Brochu, T., and Bridson, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4): 2472-2493.

Bruyns, C.D., and Senger, S. 2001. Interactive cutting of 3D surface meshes. *Computers & Graphics*, 25(4): 635-642.

Bruyns, C.D., Senger, S., Menon, A., Montgomery, K., Wildermuth, S., and Boyle, R. 2002. A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools‡. *The journal of visualization and computer animation*, 13(1): 21-42.

Carlson, M., Mucha, P.J., and Turk, G. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. *In ACM Transactions on Graphics (TOG)*, 23(3): 377-384.

Chen, H., and Sun, H. 2002. Real-time haptic sculpting in virtual volume space. *In Proceedings of the ACM symposium on Virtual reality software and technology*, 81-88.

Chen, J.X., Wechsler, H., Pullen, J.M., Zhu, Y., and MacMahon, E.B. 2001. Knee surgery assistance: patient model construction, motion simulation, and biomechanical visualization. *Biomedical Engineering, IEEE Transactions on*, 48(9): 1042-1052.

Chen, K.W., Heng, P.A., and Sun, H. 2000. Direct haptic rendering of isosurface by intermediate representation. *In Proceedings of the ACM symposium on Virtual reality software and technology*, 188-194.

Chen, M., Silver, D., Winter, A.S., Singh, V., and Cornea, N. 2003. Spatial transfer functions: a unified approach to specifying deformation in volume modelling and animation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, 35-44.

Chen, M., Correa, C., Islam, S., Jones, M.W., Shen, P.Y., Silver, D., ... and Willis, P.J. 2005. Deforming and animating discretely sampled object representations. *Eurographics 2005 STAR Reports*, 113-140.

Chen, X., Lin, Y., Wang, C., Shen, G., and Wang, X. 2012. A virtual training system using a force feedback haptic device for oral implantology. In *Transactions on Edutainment VIII*, 232-240.

Chentanez, N., Goktekin, T.G., Feldman, B.E., and O'Brien, J.F. 2006. Simultaneous coupling of fluids and deformable bodies. *In Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 83-89.

Chentanez, N., Alterovitz, R., Ritchie, D., Cho, L., Hauser, K.K., Goldberg, K., ... and O'Brien, J.F. 2009. Interactive simulation of surgical needle insertion and steering, 28(3): 88.

Chentanez, N., and Müller, M. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. *In ACM Transactions on Graphics (TOG)*, 30(4): 82.

Chmarra, M.K., Dankelman, J., van den Dobbelsteen, J.J., and Jansen, F.W. 2008. Force feedback and basic laparoscopic skills. *Surgical endoscopy*, 22(10): 2140-2148.

Choi, C., Kim, J., Han, H., Ahn, B., and Kim, J. 2009. Graphic and haptic modelling of the oesophagus for VR‐based medical simulation. *The International Journal of Medical Robotics and Computer Assisted Surgery*,5(3): 257-266.

Choi, K., Jo, S., Lee, H., and Jeong, C. 2012. CPU-based speed acceleration techniques for shear warp volume rendering. *Multimedia Tools and Applications*, 1-21.

Choi, K. S., Soo, S., and Chung, F. L. 2009. A virtual training simulator for learning cataract surgery with phacoemulsification. *Computers in biology and medicine*, 39(11): 1020-1031.

Cirbus, J. and Podhoranyi, M. 2011. Cellular automata for earth surface flow simulation.

Coles, T.R., Meglan, D., and John, N.W. 2011. The role of haptics in medical training simulators: a survey of the state of the art. *Haptics, IEEE Transactions on*, 4(1): 51-66.

Correa, C.D., and Silver, D. 2007. Programmable shaders for deformation rendering. In *SIGGRAPH/EUROGRAPHICS Conference On Graphics Hardware: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 4(5): 89-96.

Correa, C.D., Silver, D., and Chen, M. 2006. Feature aligned volume manipulation for illustration and visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5): 1069-1076.

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

Cotin, S., Duriez, C., Lenoir, J., Neumann, P., and Dawson, S. 2005. New approaches to catheter navigation for interventional radiology simulation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI* 2005: 534-542.

Coulthard, T.J., Hicks, D.M., and Van De Wiel, M.J. 2007. *Cellular modelling of river catchments and reaches: Advantages, limitations and prospects. Geomorphology*, 90(3): 192-207.

Coutinho, B.B.S., Giraldi, G., Apolinario, A., and Rodrigues, P. 2008. GPU surface flow simulation and multiresolution animation in digital terrain models.

Cover, S.A., Ezquerra, N.F., O'Brien, J.F., Rowe, R., Gadacz, T., and Palm, E. 1993. Interactively deformable models for surgery simulation. *Computer Graphics and Applications, IEEE*, 13(6): 68-75.

Crane, K., Llamas, I., and Tariq, S. 2007. Real-time simulation and rendering of 3D fluids. GPU Gems, 3(1).

Cummins, S.J., and Rudman, M. 1999. An SPH projection method. *Journal of computational physics*, 152(2): 584-607.

Daenzer, S., Montgomery, K., Dillmann, R., and Unterhinninghofen, R. 2007. Real-time smoke and bleeding simulation in virtual surgery. *In Medicine meets virtual reality*, 15: 94.

De, S., Lim, Y.J., Manivannan, M., and Srinivasan, M.A. 2006. Physically realistic virtual surgery using the point-associated finite field (PAFF) approach. *Presence: Teleoperators and Virtual Environments*, 15(3): 294-308.

Dräger, C., 2005. A ChainMail Algorithm for Direct Volume Deformation in Virtual Endoscopy Applications.

Dunkin, B., Adrales, G.L., Apelgren, K., and Mellinger, J.D. 2007. Surgical simulation: a current review. *Surgical endoscopy*, 21(3): 357-366.

Faeth, A., and Harding, C. 2009. Supporting interactive haptic shaping of 3D geologic surfaces with deformation property painting. *In Eurographics 2009-Areas Papers*, 11-17.

Fager, P.J., and von Wowern, P. 2004. The use of haptics in medical applications. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 1(1): 36-42.

Faure, F., Gilles, B., Bousquet, G., and Pai, D.K. 2011. Sparse meshless models of complex deformable solids. *ACM Transactions on Graphics (TOG)*, 30(4): 73.

Fedkiw, R., Stam, J., and Jensen, H.W. 2001. Visual simulation of smoke. *In Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 15-22.

Foster, N., and Metaxas, D. 1996. Realistic animation of liquids. *Graphical models and image processing*, 58(5): 471-483

Foster, N., and Fedkiw, R. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 23-30. ACM.

Fortmeier, D., Mastmeyer, A., and Handels, H. 2012. GPU-based visualization of deformable volumetric soft-tissue for real-time simulation of haptic needle insertion. *In Bildverarbeitung für die Medizin*, 117-122.

Fortmeier, D., Mastmeyer, A., and Handels, H. 2013. Image-based soft tissue deformation algorithms for real-time simulation of liver puncture. *Current Medical Imaging Reviews*, 9(2): 154-165.

Fortmeier, D., Mastmeyer, A., and Handels, H. 2013. Image-based palpation simulation with soft tissue deformations using chainmail on the GPU. *In Bildverarbeitung für die Medizin*, 140-145.

Forest, C., Delingette, H., and Ayache, N. 2005. Removing tetrahedra from manifold tetrahedralisation: application to real-time surgical simulation. *Medical Image Analysis*, 9(2): 113-122.

Frisken-Gibson, S.F. 1999. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4): 333-348.

Gasson, P., Lapeer, R.J., and Linney, A.D. 2004. Modelling techniques for enhanced realism in an open surgery simulation. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*: 73-78.

Georgii, J., and Westermann, R. 2006. A multi-grid framework for real-time simulation of deformable bodies. *Computers & Graphics*, 30(3): 408-415.

Gibson, S.F. 1997. 3D chainmail: a fast algorithm for deforming volumetric objects. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, 149-ff.

Gibson, S.F. 1999. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4): 333-348.

Gilles, B., Bousquet, G., Faure, F., and Pai, D.K. 2011. Frame-based elastic models. *ACM Transactions on Graphics (TOG)*, 30(2): 15.

Goksel, O., Sapchuk, K., and Salcudean, S.E. 2011. Haptic simulator for prostate brachytherapy with simulated needle and probe interaction. *Haptics, IEEE Transactions on*, 4(3): 188-198.

Golden Co. 2008. HealthGrades Quality Study: Fifth Annual Patient Safety in American Hospitals Study.

Gurusamy, K.S., Rajesh, A., Latha, P., and Brian R.D. 2009. Virtual reality training for surgical trainees in laparoscopic surgery. *Cochrane Database Syst Rev* 1, no. 4.

Halic, T., Sankaranarayanan, G., and De, S. 2010. GPU‑based efficient realistic techniques for bleeding and smoke generation in surgical simulators. *The International Journal of Medical Robotics and Computer Assisted Surgery*,6(4): 431-443.

Haque, S., and Srinivasan, S. 2006. A meta-analysis of the training effectiveness of virtual reality surgical simulators. *Information Technology in Biomedicine, IEEE Transactions on*, 10(1): 51-58.

Harlow, F.H., and Welch, J.E. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 8: 2182.

Harris, M. 2008. Cuda fluid simulation in nvidia physx. *Siggraph Asia*, 77-84.

Hart, R., and Karthigasu, K. 2007. The benefits of virtual reality simulator training for laparoscopic surgery. *Current Opinion in Obstetrics and Gynecology*, 19(4): 297-302.

Hauser, K., Alterovitz, R., Chentanez, N., Okamura, A., and Goldberg, K. 2009. Feedback control for steering needles through 3D deformable tissue using helical paths. *Robotics science and systems: online proceedings*, 37.

Herrell, S.D., and Smith, J.A. 2005. Robotic-assisted laparoscopic prostatectomy: what is the learning curve?. *Urology*, 66(5): 105-107.

Huy Viet, H.Q., Kamada, T., and Tanaka, H.T. 2006. An algorithm for cutting 3D surface meshes. *In Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 4: 762-765.

Judice, S.F., Barcellos, B., and Giraldi, G.A. 2008. A cellular automata framework for real time fluid animation. *In Proceedings of the Brazilian Symposium on Computer Games and Digital Entertainment*, 169-176.

Kass, M., and Miller, G. 1990. Rapid, stable fluid dynamics for computer graphics. *ACM SIGGRAPH Computer Graphics*, 24(4): 49-57.

Kaufmann, P., Martin, S., Botsch, M., Grinspun, E., and Gross, M. 2009a. Enrichment textures for detailed cutting of shells. In *ACM Transactions on Graphics (TOG)*, 28(3): 50. ACM.

Kaufmann, P., Martin, S., Botsch, M., and Gross, M. 2009b. Flexible simulation of deformable models using discontinuous Galerkin FEM. *Graphical Models*, 71(4): 153-167.

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

Kerwin, T., Shen, H.W., and Stredney, D. 2009. Enhancing realism of wet surfaces in temporal bone surgical simulation. *Visualization and Computer Graphics, IEEE Transactions on*, 15(5): 747-758.

Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J. 2007. Simulation of bubbles in foam with the volume control method. *In ACM Transactions on Graphics (TOG)*, 26(3): 98.

Kim, T., Thürey, N., James, D., and Gross, M. 2008. Wavelet turbulence for fluid simulation. *In ACM Transactions on Graphics (TOG)*, 27(3): 50.

Kim, S., Park, J., and Park, J. 2010. Progressive mesh cutting for real-time haptic incision simulator. *In ACM SIGGRAPH ASIA 2010 Posters*, 61.

Kobayashi, Y., Onishi, A., Watanabe, H., Hoshi, T., Kawamura, K., Hashizume, M., and Fujie, M.G. 2010. Development of an integrated needle insertion system with image guidance and deformation simulation. *Computerized Medical Imaging and Graphics*, 34(1): 9-18.

Kohn, L.T., Corrigan, J.M., and Donaldson, M.S. (Eds.). 2000. To err is human: building a safer health system, 627. National Academies Press.

Krog, Ø.E., and Elster, A.C. 2012. Fast gpu-based fluid simulations using sph. *In Applied Parallel and Scientific Computing*, 98-109.

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

Lefebvre, S., Hornus, S., and Neyret, F. 2005. GPU Gems 2. chapter 37: Octree Textures on the GPU.

Li, W.C., Levy, B., and Paul, J.C. 2005. Mesh editing with an embedded network of curves. In *Shape Modeling and Applications, 2005 International Conference*, 62-71. IEEE.

Li, X., Gu, L., Zhang, S., Zhang, J., Zheng, G., Huang, P., and Xu, J. 2008. Hierarchical spatial hashing-based collision detection and hybrid collision response in a haptic surgery simulator. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 4(1): 77-86.

Li, Y., and Brodlie, K. 2003. Soft object modelling with generalised chainmail - extending the boundaries of web-based graphics. In *Computer Graphics Forum*, 22(4): 717-727. Blackwell Publishing, Inc.

Lim, Y.J., Hu, J., Chang, C.Y., and Tardella, N. 2006. Soft tissue deformation and cutting simulation for the multimodal surgery training. In *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on*, 635-640. IEEE.

Lim, Y.J., Jin, W., and De, S. 2007. On some recent advances in multimodal surgery simulation: A hybrid approach to surgical cutting and the use of video images for enhanced realism. *Presence: Teleoperators and Virtual Environments*, 16(6): 563-583.

Lin, S., Lee, Y.S., and Narayan, R.J. 2007. Snapping algorithm and heterogeneous bio-tissues modelling for medical surgical simulation and product prototyping. *Virtual and Physical Prototyping*, 2(2): 89-101.

Lin, S.Y., Lee, Y.S. and Narayan, R. 2007. Snapping algorithm and heterogeneous bio-tissues modelling for medical surgical simulation and product prototyping. *Virtual and Physical Prototyping*, 2(2): 89-101.

Liu, A., Kaufmann, C., and Tanaka, D. 2001a. An architecture for simulating needle-based surgical procedures. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI* 2001: 1137-1144.

Liu, A., Kaufmann, C., and Ritchie, T. 2001b. A computer-based simulator for diagnostic peritoneal lavage. *Studies in health technology and informatics*: 279-285.

Liu, A., Tendick, F., Cleary, K., and Kaufmann, C. 2003. A survey of surgical simulation: applications, technology, and education. *Presence: Teleoperators & Virtual Environments*, 12(6): 599-614.

Liu, Y., and Pender, G. 2010. A new rapid flood inundation model. *In Proceedings of the first IAHR European Congress*, 4-6.

Lin, S.Y., Narayan, R., and Lee, Y.S. 2008. Heterogeneous deformable modelling and topology modification for surgical cutting simulation with haptic interfaces. *Computer-Aided Design and Applications*, 5(6): 877-888.

Lorensen, W.E., and Cline, H.E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics* 21(4): 163-169.

Lundin, K.E., Sillen, M., Cooper, M.D., and Ynnerman, A. 2005. Haptic visualization of computational fluid dynamics data using reactive forces. *In Electronic Imaging 2005*, 31-41.

Malone, H.R., Syed, O.N., Downes, M.S., D'Ambrosio, A.L., Quest, D.O., and Kaiser, M.G. 2010. Simulation in neurosurgery: a review of computer-based simulation environments and their surgical applications. *Neurosurgery,* 67(4): 1105-1116.

Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M., and Taylor II, R.M. 1996. Adding force feedback to graphics systems: Issues and solutions. *In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 447-452.

Marroquim, R., Maximo, A., Farias, R., and Esperança, C. 2008. Volume and isosurface rendering with GPU-accelerated cell projection. *In Computer Graphics Forum*, 27(1): 24-35.

Masutani, Y., Inoue, Y., Ishii, K., Kumai, N., Kimura, F., and Sakuma, I. 2004. Development of surgical simulator based on FEM and deformable volume-rendering. *In Medical Imaging 2004*, 500-507. International Society for Optics and Photonics.

Maximo, A., Marroquim, R., and Farias, R. 2010. Hardware-assisted projected tetrahedra. *In Computer Graphics Forum*, 29(3): 903-912.

Mayooran, Z., Watterson, L., Withers, P., Line, J., Arnett, W., and Horley, R. 2006. Mediseus epidural: full-procedure training simulator for epidural analgesia in labour. *In SimTecT Healthcare Simulation Conference*.

Mazura, A., and Seifert, S. 1997. Virtual cutting in medical data. *Studies in Health Technology and Informatics*, 420-429.

Mei, X., Decaudin, P., and Hu, B.G. 2007. Fast hydraulic erosion simulation and visualization on GPU. *In Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, 47-56.

Mesit, J., Guha, R.K., and Furlong, W.P. 2010. Simulation of lung respiration function using soft body model. In *Computer Modelling and Simulation (EMS), 2010 Fourth UKSim European Symposium on*: 102-107. IEEE.

Misra, S., Ramesh, K.T., and Okamura, A.M. 2008. Modelling of tool-tissue interactions for computer-based surgical simulation: a literature review.

Monaghan, J.J. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30: 543-574.

Mor, A.B., and Kanade, T. 2000. Modifying soft tissue models: Progressive cutting with minimal new element creation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2000*, 598-607. Springer Berlin Heidelberg.

Mosegaard, J., Herborg, P., and Sorensen, T.S. 2005. A GPU accelerated spring mass system for surgical simulation. *Studies in health technology and informatics*, 111: 342-348.

Mosegaard, J. 2006. Cardiac Surgery Simulation-Graphics Hardware meets Congenital Heart Disease (Doctoral dissertation, Ph. D. thesis, Department of Computer Science, University of Aarhus, Denmark).

Müller, M., and Chentanez, N. 2011. Solid simulation with oriented particles. *In ACM Transactions on Graphics (TOG),* 30(4): 92.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. 2005. Meshless deformations based on shape matching. In *ACM Transactions on Graphics (TOG),* 24(3): 471-478.

Müller, M., Schirm, S., and Teschner, M. 2004. Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics. *Technology and Health Care*, 12(1): 25-31.

Nakao, M. 2003. Cardiac surgery simulation with active interaction and adaptive physics-based Modelling. USA: *Department of Social Infor-matics, Graduate School of Informatics, Kyoto University*.

Nakao, M., Hung, K.W.C., Yano, S., Yoshimura, K., and Minato, K. 2010. Adaptive proxy geometry for direct volume manipulation. *In Pacific Visualization Symposium (PacificVis),* 161-168.

Nealen, A., Müller, M., Keiser, R., Boxerman, E., and Carlson, M. 2006. Physically based deformable models in computer graphics. In *Computer Graphics Forum,* 25(4): 809-836. Blackwell Publishing Ltd.

Neyret, F., Heiss, R., and Sénégas, F. 2002. Realistic rendering of an organ surface in real-time for laparoscopic surgery simulation. *The Visual Computer*, 18(3): 135-149.

Nienhuys, H.W., and van der Stappen, A.F. 2001. A surgery simulation supporting cuts and finite element deformation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2001*, 145-152.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., and Purcell, T.J. 2007. A Survey of General‐Purpose Computation on Graphics Hardware. *In Computer graphics forum,* 26(1): 80-113.

Palmerius, K.L. 2007. Fast and high precision volume haptics. *In EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007.* Second Joint, 501-506.

Pang, W.M., Qin, J., Chui, Y.P., Wong, T.T., Leung, K.S., and Heng, P.A. 2007. Orthopedics surgery trainer with PPU-accelerated blood and tissue simulation. *In Medical Image Computing and Computer-Assisted Intervention–MICCAI,* 842-849.

Pan, J. J., Chang, J., Yang, X., Zhang, J. J., Qureshi, T., Howell, R., and Hickish, T. 2011. Graphic and haptic simulation system for virtual laparoscopic rectum surgery. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 7(3): 304-317.

Pao, L.Y., and Lawrence, D.A. 1998. Synergistic visual/haptic computer interfaces. *In Proc. of Japan/USA/Vietnam Workshop on Research and Education in Systems, Computation, and Control Engineering,* 155-162.

Park, J.S., Chung, M.S., Hwang, S.B., Lee, Y.S., Har, D.H., and Park, H.S. 2005. Visible Korean human: improved serially sectioned images of the entire body. *Medical Imaging, IEEE Transactions on*, 24(3): 352-360.

Peters, J. H., Fried, G.M., Swanstrom, L.L., Soper, N.J., Sillin, L.F., Schirmer, B., and Hoffman, K. 2004. Development and validation of a comprehensive program of education and assessment of the basic fundamentals of laparoscopic surgery. *Surgery,* 135(1): 21-27.

Phenomena, A. N. 2004. SDK White Paper.

Picod, G., Jambon, A.C., Vinatier, D., and Dubois, P. 2005. What can the operator actually feel when performing a laparoscopy?. *Surgical Endoscopy and Other Interventional Techniques*, 19(1): 95-100.

Premžoe, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R.T. 2003. Particle-Based Simulation of Fluids. In *Computer Graphics Forum*, 22(3): 401-410. Blackwell Publishing, Inc.

Qin, J., Pang, W.M., Chui, Y.P., Xie, Y.M., Wong, T.T., Poon, W. S., ... and Heng, P.A. 2007. Hardware-accelerated bleeding simulation for virtual surgery. *In MICCAI 2007 Workshop Proceedings*, 133.

Qin, J., Choi, K.S., Poon, W.S., and Heng, P.A. 2009. A framework using cluster-based hybrid network architecture for collaborative virtual surgery. *Computer methods and programs in biomedicine*, 96(3): 205-216.

Qin, J., Pang, W.M., Nguyen, B.P., Ni, D., and Chui, C.K. 2010. Particle-based simulation of blood flow and vessel wall interactions in virtual surgery. *In Proceedings of the 2010 Symposium on Information and Communication Technology*, 128-133.

Ranal, M.A., Setan, H., Majid, Z., and Chong, A.K. 2007. Simulation of interactive cutting tool for craniofacial osteotomy planning. In *Computer Graphics, Imaging and Visualisation, 2007. CGIV'07*, 506-512. IEEE.

Reeves, W.T. 1983. Particle systems - a technique for modelling a class of fuzzy objects. In *ACM SIGGRAPH Computer Graphics* ,17(3): 359-375. ACM.

Rianto, S., and Li, L. 2010. Fluid dynamic visualisations of cuttings-bleeding for virtual reality heart beating surgery simulation. *In Proceedings of the Thirty-Third Australasian Conferenc on Computer Science*, 102: 53-60.

Richa, R., Poignet, P., and Liu, C. (2010). Three-dimensional motion tracking for beating heart surgery using a thin-plate spline deformable model. *The International Journal of Robotics Research*, 29(2-3): 218-230.

Rivers, A.R., and James, D.L. 2007. FastLSM: fast lattice shape matching for robust real-time deformation. *In ACM Transactions on Graphics (TOG)*, 26(3): 82.

Robles-De-La-Torre, G. 2006. The importance of the sense of touch in virtual and real environments. *Multimedia, IEEE*, 13(3): 24-30.

Ruspini, D.C., Kolarov, K., and Khatib, O. 1997. The haptic display of complex graphical environments. *In Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 345-352.

Sathappan, O.L., Chitra, P., Venkatesh, P., and Prabhu, M. 2011. Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system. *International Journal of Information Technology, Communications and Convergence*, 1(2): 146-158.

Santhanam, A.P., Imielinska, C., Davenport, P., Kupelian, P., and Rolland, J.P. 2008. Modelling real-time 3-D lung deformations for medical visualization. *Information Technology in Biomedicine, IEEE Transactions on*, 12(2): 257-270.

Schiff, J.L. 2011. Cellular automata: a discrete view of the world, 45.

Schill, M.A., Gibson, S.F., Bender, H.J., and Männer, R. 1998. Biomechanical simulation of the vitreous humor in the eye using an enhanced chainmail algorithm. In *Medical Image Computing and Computer-Assisted Interventation—MICCAI'98*, 679-687.

Schulze, F., Bühler, K., and Hadwiger, M. 2007. Interactive deformation and visualization of large volume datasets. *In Proceedings of International Conference on Computer Graphics Theory and Applications*, 39-46.

Schulze, F., Bühler, K., and Hadwiger, M. 2009. Direct volume deformation. *In Computer Vision and Computer Graphics. Theory and Applications*, 59-72.

Schulze, F. 2006. Direkte Deformation von Volumendaten zur Simulation von Weichgewebe in der Volume Rendering basierten Operationssimulation.

Newman, T.S., and Yi, H. 2006. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5): 854-879.

Schutyser, F., Van Cleynenbreugel, J., Nadjmi, N., Schoenaers, J., and Suetens, P. 2000. 3D image-based planning for unilateral mandibular distraction.

Sela, G., Schein, S., and Elber, G. 2004. Real-time incision simulation using discontinuous free form deformation. *In Medical Simulation* , 114-123.

Sela, G., Subag, J., Lindblad, A., Albocher, D., Schein, S., and Elber, G. 2007. Real-time haptic incision simulation using FEM-based discontinuous free-form deformation. *Computer-Aided Design*, 39(8): 685-693.

Selle, A., Lentine, M., and Fedkiw, R. 2008. A mass spring model for hair simulation. In *ACM Transactions on Graphics (TOG),* 27(3): 64. ACM.

Serby, D., Harders, M., and Székely, G. 2001. A new approach to cutting into finite element models. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2001*, 425-433. Springer Berlin Heidelberg.

Seymour, N.E. 2008. VR to OR: a review of the evidence that virtual reality simulation improves operating room performance. *World Journal of Surgery*, 32(2): 182-188.

Shirley, P., and Tuchman, A. 1990. A polygonal approximation to direct scalar volume rendering, 24(5): 63-70.

Singh, V., and Silver, D. 2004. Interactive volume manipulation with selective rendering for improved visualization. *In Volume Visualization and Graphics, 2004 IEEE Symposium*, 95-102.

Singh, V., Silver, D., and Cornea, N. 2003. Real-time volume manipulation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, 45-51. ACM.

Solenthaler, B., and Pajarola, R. 2009. Predictive-corrective incompressible SPH. *In ACM Transactions on Graphics (TOG)*,28(3): 40.

Sørensen, M.S., Dobrzeniecki, A.B., Larsen, P., Frisch, T., Sporring, J., and Darvann, T. A. 2002. The visible ear: a digital image library of the temporal bone. *ORL*, *64*(6): 378-381.

Stam, J., and Fiume, E. 1993. Turbulent wind fields for gaseous phenomena. *In Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 369-376.

Stam, J., and Fiume, E. 1995. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques,* 129-136. ACM.

Stam, J. 1999. Stable fluids. *In Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121-128.

Stam, J. 2003. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, 18.

Surendran, D., Purusothaman, T., and Balachandar, R.A. 2011. A generic interface for resource aggregation in grid of grids. *International Journal of Information Technology, Communications and Convergence*, 1(2): 159-172.

Sweet, R., Porter, J., Oppenheimer, P., Hendrickson, D., Gupta, A., and Weghorst, S. 2002. Third prize: simulation of bleeding in endoscopic procedures using virtual reality. *Journal of endourology*, 16(7): 451-455.

Tahmasebi, A.M., Hashtrudi-Zaad, K., Thompson, D., and Abolmaesumi, P. 2008. A framework for the design of a novel haptic-based medical training simulator. *Information Technology in Biomedicine, IEEE Transactions on*,12(5): 658-666.

Takeshita, D., Ota, S., Tamura, M., Fujimoto, T., Muraoka, K., and Chiba, N. 2003. Particle-based visual simulation of explosive flames. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference*, 482-486. IEEE.

Tasto, J.L., Nguyen, B.H., Cunningham, R., and Merril, G.L. (1999). CathSim™: An intravascular catheterization simulator on a PC. *Medicine Meets Virtual Reality, 8: The Convergence of Physical and Informational Technologies: Options for a New Era in Health Care*, 62: 360.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. 1987. Elastically deformable models. In *ACM Siggraph Computer Graphics* 21(4): 205-214.

Terzopoulos, D., and Fleischer, K. 1988. Deformable models. *The Visual Computer*, 4(6): 306-331.

Terzopoulos, D., and Witkin, A. 1988. Physically based models with rigid and deformable components. *Computer Graphics and Applications, IEEE*, 8(6): 41-51.

Terzopoulos, D., Witkin, A., and Kass, M. 1988. Constraints on deformable models: recovering 3D shape and non-rigid motion. *Artificial intelligence*, 36(1): 91-123.

Teschner, M., Stefan K., Bruno H., Gabriel Z., Laks R., Arnulph F., M-P. Cani *et al.* 2005. Collision detection for deformable objects. *In Computer Graphics Forum*, 24(1): 61-81.

Thompson, J.R., Leonard, A.C., Doarn, C.R., Roesch, M.J., and Broderick, T.J. 2011. Limited value of haptics in virtual reality laparoscopic cholecystectomy training. *Surgical endoscopy*, 25(4): 1107-1114.

Turkiyyah, G.M., Karam, W.B., Ajami, Z., and Nasri, A. 2011. Mesh cutting during real-time physical simulation. *Computer-Aided Design*, 43(7): 809-819.

Van Dongen, K.W., Tournoij, E., Van der Zee, D.C., Schijven, M.P., and Broeders, I. A.M.J. 2007. Construct validity of the LapSim: can the LapSim virtual reality simulator distinguish between novices and experts?. *Surgical endoscopy*, 21(8): 1413-1417.

Van der Meijden, O.A.J., and Schijven, M.P. 2009. The value of haptic feedback in conventional and robot-assisted minimal invasive surgery and virtual reality training: a current review. *Surgical endoscopy*, 23(6): 1180-1190.

Vickers, A.J., Caroline J.S., Marcel, H., Ingolf, T., Philippe, K., Luis, M., Gunther, J., and Bertrand G. 2009. The surgical learning curve for laparoscopic compared to open radical prostatectomy: a retrospective cohort study. *The lancet oncology* 10(5): 475.

Wang, H.Y., Wang, Y., and Esen, H. 2009. Modelling of deformable objects in haptic rendering system for virtual reality. *Proceedings of the 2009 IEEE International Conference on Mechatronics and Automation*, 90-94.

Wang, P., Becker, A.A., Jones, I.A., Glover, A.T., Benford, S.D., Greenhalgh, C.M., and Vloeberghs, M. 2006. A virtual reality surgery simulation of cutting and retraction in neurosurgery with force-feedback. *Computer methods and programs in biomedicine*, 84(1): 11-18.

Webster, R.W., Zimmerman, D.I., Mohler, B.J., Melkonian, M.G., and Haluck, R.S. 2001. A prototype haptic suturing simulator. *Studies in health technology and informatics*, 81: 567.

Westebring-van der Putten, E.P., Goossens, R.H.M., Jakimowicz, J.J., and Dankelman, J. 2008. Haptics in minimally invasive surgery-a review. *Minimally Invasive Therapy & Allied Technologies*, 17(1): 3-16.

Westermann, R., and Rezk-Salama, C. 2001. Real-Time Volume Deformations. *In Computer Graphics Forum*, 20(3): 443-451.

Wilson, M.S., Middlebrook, A., Sutton, C., Stone, R., and McCloy, R.F. 1997. MIST VR: a virtual reality trainer for laparoscopic surgery assesses performance. *Annals of the Royal College of Surgeons of England*, 79(6): 403.

Wittek, A., Dutta-Roy, T., Taylor, Z., Horton, A., Washio, T., Chinzei, K., and Miller, K. 2008. Subject-specific non-linear biomechanical model of needle insertion into brain. *Computer methods in biomechanics and biomedical engineering*, 11(2): 135-146.

Wong, K.C.H., Siu, T.Y.H., Heng, P.A., and Sun, H. 1998. Interactive volume cutting. In *Graphics Interface*, 99-106. CANADIAN INFORMATION PROCESSING SOCIETY.

Xu, Q., Gledbill, D., and Xu, Z. 2011. Volume deformation based on model-fitting surface extraction. *In Automation and Computing (ICAC)* 17th *International Conference on*, 161-166.

Zátonyi, J., Paget, R., Székely, G., Grassi, M., and Bajka, M. 2005. Real-time synthesis of bleeding for virtual hysteroscopy. *Medical image analysis*, 9(3): 255-266.

Zhang, S.X., Heng, P.A., Liu, Z.J., Tan, L.W., Qiu, M.G., Li, Q.Y., ... and Xie, Y.M. 2003. Creation of the Chinese visible human dataset. *The Anatomical Record Part B: The New Anatomist*, 275(1): 190-195.

Zhang, H., Payandeh, S., and Dill, J. 2004. On cutting and dissection of virtual deformable objects. *In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 4: 3908-3913.

Zhang, Y., Solenthaler, B., and Pajarola, R. 2008. Adaptive sampling and rendering of fluids on the GPU. *In Proceedings of the Fifth Eurographics/IEEE VGTC conference on Point-Based Graphics*, 137-146.

Zhang, J., Gu, L., Li, X., and Fang, M. 2009. An advanced hybrid cutting method with an improved state machine for surgical simulation. *Computerized Medical Imaging and Graphics*, 33(1): 63-71.

Zhang, F., Hu, L., Wu, J., and Shen, X. 2011. A SPH-based method for interactive fluids simulation on the multi-GPU. *In Proceedings of the 10th international conference on virtual reality continuum and its applications in industry*, 423-426.

Zhang, Q., Peters, T.M., and Eagleson, R. 2011. Medical image volumetric visualization: algorithms, pipelines, and surgical applications. *In Medical Image Processing* , 291-317.

Zhu, B., Gu, L., and Zhou, Z. 2010. Particle-based deformable modelling with pre-computed surface data in real-time surgical simulation. *In Medical Imaging and Augmented Reality*, 503-512.

Zhu, Y., Magee, D., Ratnalingam, R., and Kessel, D. 2007. A training system for ultrasound-guided needle insertion procedures. In *Medical Image Computing* *and Computer-Assisted Intervention–MICCAI 2007*, 566-574. Springer Berlin Heidelberg.