Interactive Soft Tissue for Surgical Simulation

by

Gregory S. Ruthenbeck

BEng(Hons)

Doctorate of Philosophy in Engineering in the School of Computer Science, Engineering and Mathematics at Flinders University, South Australia.

Abstract

Interactive Soft Tissue for Surgical Simulation

By, Gregory S. Ruthenbeck

Doctorate of Philosophy in Engineering at Flinders University of South Australia.

Principal supervisor: Prof. Karen Reynolds. Co-supervisor: Assoc. Prof. Paul Calder.

Medical simulation has the potential to revolutionise the training of medical practitioners. Advantages include reduced risk to patients, increased access to rare scenarios and virtually unlimited repeatability. However, in order to fulfil its potential, medical simulators require techniques to provide realistic user interaction with the simulated patient. Specifically, compelling real-time simulations that allow the trainee to interact with and modify tissues, as if they were practising on real patients.

A key challenge when simulating interactive tissue is reducing the computational processing required to simulate the mechanical behaviour. One successful method of increasing the visual fidelity of deformable models while limiting the complexity of the mechanical simulation is to bind a coarse mechanical simulation to a more detailed shell mesh. But even with reduced complexity, the processing required for real-time interactive mechanical simulation often limits the fidelity of the medical simulation overall. With recent advances in the programmability and processing power of massively parallel processors such as graphics processing units (GPUs), suitably designed algorithms can achieve significant improvements in performance.

This thesis describes an ablatable soft-tissue simulation framework, a new approach to interactive mechanical simulation for virtual reality (VR) surgical training simulators that makes efficient use of parallel hardware to deliver a realistic and versatile interactive real-time soft tissue simulation for use in medical simulators.

Acknowledgments

This PhD was undertaken with financial support from an Australian Research Council APAI scholarship with sponsorship from Medical Realities Pty Ltd.

I would especially like to thank Prof. Karen Reynolds, firstly for making this research possible, and secondly for her unerring support and counsel during my candidature. I would also like to thank A.Prof. Paul Calder for his direction and open minded critical input. Most importantly, thanks to my loving wife Riikka and my son Isaac for giving me perspective and all the important things.

Cheers too to Fabian, Cyle, Dave, Katie, Martin, Lynne, and Pete for chats, cheers, and Chingas chillies that kept me sane.

Declaration

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signed:

Date:

Contents

Figures

Glossary

Ablation The volumetric removal of tissue.

API Application Programming Interface. (Software library.)

BEM Boundary Element Method of mechanical simulation.

CT Computed Tomography medical imaging.

CRMS The Cubic Rotational Mechanical Simulation described in Ch. 5.

CUDA Nvidia's Compute Unified Device Architecture API for GPUs.

DirectX Microsoft's real time 3D graphics API.

ENT The field of medicine concerned with the Ear, Nose and Throat.

ESS Endoscopic Sinus Surgery.

FEM Finite Element Method of mechanical simulation.

FLOPs A measure of processing performance. Floating Point Operations per second.

FPGA Fully Programmable Gate Array. A type of programmable computing hardware.

Fragment Shader Pixel Shaders and Vertex Shaders are two types of Fragment Shaders. A small program that is part of a graphics pipeline.

GPGPU General Purpose Graphics Processing Unit.

GPU Graphics Processing Unit. A piece of computing hardware used to perform processing to generate interactive 3D graphics.

Haptics The field related to precision force feedback (tactile feedback).

IMT Interactive Marching Cubes (describe in Chapter 6).

ISim An Endotracheal Intubation Simulation (described in Chapter 9).

Iso-surface When a volume contains a 3D grid of density values, an iso-surface is an interpolated surface that connects locations of the same density.

MIS Minimally Invasive Surgery (e.g. key-hole surgery).

MRI Magnetic Resonance medical Imaging.

NURBS Non Uniform Rotational Bezier Splines.

Pixel Shader A program that defines the lighting algorithm used to compute the colour of a given pixel in a rendered 3D scene.

Polytopes Types of polygonal elements (e.g. triangles, squares (quads), etc).

Quaternion A mathematical method commonly used to represent rotations.

Rasterize The process of converting polytopes into pixels.

Shader A shading algorithm.

SIMD Single Instruction, Multiple Data

SPMD Single Program, Multiple Data

SOFA Simulation Open Framework Architecture.

TSF The Tissue Simulation Framework that is the subject of this thesis.

Vertex Shader A GPU program that maps vertex (point) locations to screen-space (pixel) locations.

Voxel A volumetric pixel. A quantum of a regular cubic grid.

Chapter 1. Introduction

Unlike interactive computer entertainment, the key interactions within virtual reality (VR) surgical simulations do not simply move a vehicle, aircraft or point of view. Surgical simulations must allow the user to perform subtle interactions with simulated tissues in a realistic manner. Even though many of the requisite technologies have matured to a level that supports the degree of realism required, new techniques are required to enable interactive mechanical simulation of tissues and organs with realistic tactile feedback. Consequently, new medical simulations are expensive to develop and leave little time to focus on overarching requirements like content, scenarios, and learning outcomes.

In traditional teaching, opportunities to practice skills are often limited by access to willing patients or cadavers. Furthermore, practise of any non-expert on patients exposes these patients to increased risk. Hence, there is great potential for medical simulators to improve medical training and reduce the training system's reliance on early skills development on patients. Immersion is an important part of the simulation-based learning experience. Contextual learning facilitates recall of the skills practised and hence improves learning outcomes. However, realism needs more than visual or auditory effects; tactile feedback is critical in a large number of medical interventions.

Delivering a realistic and compelling manual interaction requires a user interface that replicates the key interactions that normally occur with a real patient. Relatively recent advances in computer interfaces have produced devices that accurately capture the motion of a stylus in three dimensions. A number of these devices also deliver precise force feedback. These haptic devices provide the hardware required to develop a new range of medical simulations with the ability to accurately simulate the key manual interaction. However, just as a computer display requires algorithms and rendering techniques to deliver visual realism,

haptic devices also require the development of specialised algorithms to deliver the same levels of realism to our sense of touch.

Many medical procedures involve manipulating and modifying intricate structures with diverse mechanical characteristics using subtle visual and haptic cues for guidance. For example, even the relatively simple surgical procedure performed to remove a patient's tonsils involves identifying the boundary of the tonsil and following a thin layer of separating tissue. This separating layer is identified using subtle cues involving not only the appearance, but also subtle variations in the mechanical characteristics at and around this boundary layer that influence the 'feel' encountered by the surgeon. Many other surgical procedures are even more intricate.

Developing the simulation software to achieve the required subtleties and levels of realism requires a range of technologies to work efficiently in unison. Software libraries developed for other types of simulation and computer entertainment provide a range of useful features. Rendering and visualisation alone can consume a large fraction of a simulation project's development time. Scene graph and rendering libraries can save considerable time during development via features such as managed asset loading and efficient management of the graphics pipeline for high-quality rendering. Similarly, software libraries to support common tasks such as real-time collision detection and physics-based effects are also available. Prudent use of these libraries can save time by reducing the need to re-implement common features. However, despite the apparent benefits of using these libraries, simulating tissue realistically requires systems to be efficiently integrated at a low level. Hence, very few features of existing high-level software libraries can feasibly be combined to deliver the high quality key interactions required in medical simulations.

A compelling tissue simulation must exhibit realistic mechanical behaviour in response to user interaction. Modelling mechanical behaviour at interactive rates is the subject of continuing research. One strategy for reducing run-time computations is to move as much processing offline as possible. This approach has produced some excellent results, although it does not produce a model that can be ablated or cut interactively as needed in surgical simulations. Similarly, existing

techniques that use a coarse mechanical simulation bound to a detailed visual representation result in a model that cannot be modified (cut or ablated) interactively. Hence, a new approach is required to cater to medical simulation's unique requirements.

This thesis describes a new method for simulating interactively ablatable soft-tissue with haptic feedback at higher resolution than existing methods. The method exploits the parallel computing capabilities of modern graphics processing units to achieve diverse material mechanical characteristics at higher resolution and haptically interactive rates. In order to place the new tissue simulation in context this thesis also reviews the state-of-the art of medical simulation and the relevant existing technologies.

1.1 Thesis Aims

The tissue simulation framework presented in this thesis improves the effectiveness of virtual reality medical simulations by addressing the following aims:

Aim 1: To review the current state of medical simulation and relevant developer tools.

Aim 2: To plausibly simulate the mechanical characteristics of a diverse range of tissues.

Aim 3: To enable users to manipulate (deform, cut and ablate) the simulated tissue realistically.

Aim 4: To provide accurate force feedback in response to interactions with the tissue.

Aim 5: To maximise detail and visual realism.

1.2 Thesis Outline

This section provides an overview of the remainder of this dissertation.

Chapter 2 begins with a discussion of the role of medical simulations in medical teaching. The relative advantages of simulation-based teaching over existing teaching methods are summarised. A closer focus on how simulations can further improve the learning experience of students is provided beginning with an overview of learning modalities and a discussion of how simulations can better cater to students with different learning styles. The current state of the art of VR

medical simulations is then presented. Thereby, Chapter 2 develops a deep understanding of the current state of medical simulation in partial fulfillment of Aim 1.

Chapter 3 critically reviews and summarises the software libraries and existing developer tools that are available for VR medical simulation development. Performance of real-time interactive systems is limited by the computing hardware on which it executes. Significant recent advances in parallel computing hardware have resulted in devices with substantially increased processing capabilities. Two such devices are summarised together with their significance to medical simulations.

Chapter 3 completes the discussion of existing simulations and development tools thereby addressing Aim 1. Subsequent chapters contain more specific reviews of the literature related to the development of the tissue simulation framework that is the subject of this thesis.

Chapter 4 presents the design rationale and an overview of the tissue simulation framework (Aims 2-5).

Chapter 5 addresses Aim 2 (and also relates to Aims 3 and 4). Prior work in the area of real-time mechanical simulation of soft bodies is presented. An overview of mesh topologies is given to provide context for the design decisions made. A new method of modelling deformable soft-tissues in real time is presented in which the algorithms developed allow the system to work efficiently with the other components of the tissue simulation. Specific optimisations to facilitate efficient execution on GPGPU hardware are detailed together with a number of enhancements which enable the simulation to model a diverse range of tissues with minimal impact to processing load.

Chapter 6 addresses Aims 3 and 5. It reviews current methods for polygonal surface mesh generation from volumetric data and presents a new method for creating polygonal surfaces from volumetric data that can be interactively modified.

Chapter 7 addresses Aims 2, 3 and 5. It describes how the mechanical simulation (Chapter 5) and interactive marching tetrahedra (Chapter 6) components were combined to create the tissue simulation framework.

Chapter 8 addresses Aim 4. It briefly presents the range of haptic devices currently available and reviews the available haptic rendering software libraries. It then describes three alternatives for haptic rendering of the tissue simulation. Best usage scenarios for each approach are discussed. A simple new approach for testing and presenting haptic rendering algorithms is described and used to evaluate the haptic rendering methods.

Chapter 9 describes three new medical simulators developed by the author. The simulators make use of the tissue simulation framework (described in chapters 4 to 8) and demonstrate its effectiveness when used to provide the key interaction.

Chapter 10 summarises the contributions of this thesis and identifies promising directions for future work.

Chapter 2. Virtual Reality for Medical Training

Virtual reality (VR) medical simulations will revolutionise medical teaching in the next decade [52, 84]. Traditional medical training has a number of deficiencies that simulation-based training can directly remedy. When teaching relies on developing skills by practicing on actual patients, these patients are exposed to unnecessary risks. Further, by limiting teaching to only the cases that present during their training, trainees do not receive consistent learning opportunities and may have insufficient practice of important skills. Conversely, simulations deliver a tailored learning experience that can be standardised, and can cater to different learning styles in ways that traditional teaching cannot. They also facilitate self-directed learning and allow trainees to develop skills at their own pace and to repeat specific scenarios that enable them to remedy skills deficiencies in a safe environment. This chapter explores the benefits of VR simulation-based medical training and reviews the current state of the art.

Until recently, medical training had remained largely unchanged for hundreds of years despite changes in the tools and techniques used to practice medicine [53]. Even today, patients are put at risk during normal training when skills are practiced on real patients [5, 60]. As stated by Roberts *et al*. in 2006, “Surgical training is changing: one hundred years of tradition is being challenged by legal and ethical concerns for patient safety, work hours restrictions, the cost of operating room time, and complications” [120]. Simulation offers more efficient training that, unlike traditional training, is completely repeatable. Vozenilek *et al*. neatly capture the significance and potential of simulation-based medical training by reforming the old dictum, “See one, do one, teach one” to become, “See one, simulate many, do one competently, and teach everyone” [157]. As the

effectiveness of simulation is demonstrated [3, 8, 57, 133], acceptance of simulation-based training will increase.

During the learning of any given proficiency there is increased risk to the patient [60, 168]. Simulation-based medical training reduces the risks to which patients are exposed [168]. Training institutions routinely require trainees to practice on real human patients (who are there because of their declining health). This presents a conflict of interests; what's best for the learning of the trainee isn't what's best for the patient. Some of the risk can be mitigated by ensuring that trainees are closely supervised and that each mentor is responsible for the smallest possible number of trainees. However, each trainee must perform deliberate practice to develop skills to an expert level [41]. So, rather than expose patients to serious risks, simulations offer a safe alternative where trainees can practice repeatedly to gain increased proficiency in complete safety [29].

The acquisition of clinical ward skills by undergraduate medical trainees can be haphazard [79]. Not all trainees are exposed to a complete range of situations and cases, which in turn limits the opportunities to practice and develop key skills. Additionally, trainees often work in groups where it is possible for individuals to avoid situations that confront their deficiencies. Medical emergencies in particular require quick and accurate assessment of the situation and prompt appropriate action. Simulation can be particularly effective in improving trainees' performance in medical emergencies [160] by allowing them to experience the emergency rather than simply discussing how to manage one. Simulations allow trainees to experience situations safely and repeatably, no matter how rare or dangerous these scenarios are in real life.

Just as exposure to rare situations can be limited, medical trainees may never gain experience recognising and treating rare conditions, pathologies, or responding to rare events. Simulations can readily address this by incorporating almost any symptoms, conditions, or situations into a training scenario. At the single-patient level, simulations provide a unique capacity for simulating uncommon conditions and rare surgical events. Further, simulation provides a fully controlled environment to train practitioners how to handle pressure, retain their composure, and take appropriate action. This is similar to pilots practicing how to

handle rare events such as engine fires, unresponsive controls, or any number of faults. If a trainee has been trained to handle the pressure and take appropriate action, serious complications are avoidable.

Simulations are repeatable. Simulation developers have fine-grained control of the details of each simulated training scenario. This control gives new flexibility for targeted learning, which provides a mechanism for more rapid development of training programs and simplifies adaption of training in response to any number of events, including widespread changes to treatment protocols or healthcare policies. Simulations that focus directly on these changes can be an excellent way of updating skills. Specific deficiencies of individual students may also be identified during training. Though it may be possible to tailor course content to address group deficiencies via traditional means, simulation offers new potential for individuals to benefit from self-directed learning. With procedural content and user authoring, simulations give users the ability to contribute to content thereby building and refining content available to a wide audience.

Procedural simulations provide a new mechanism for increasing independent and self-directed learning. Simulated scenarios can be repeated any number of times, and simulation users can thereby repeat difficult procedures until they have mastered them. Trainees can also perform self-directed learning, which focuses learning to address known deficiencies.

Simulation can open new channels of communication. For example, a surgical simulation provides new opportunities to simplify communication by allowing students to point to or "grab" an anatomical feature and say "what's this?" rather than using cadavers or plastic models.

Simulation reproducibility gives training institutions a means of performing fully standardised computer-based assessment. This extends to a new range of proficiencies that may be un-assessable via traditional means. For example, a simulation can record the motion of simulated surgical implements, measure unsteadiness, or detect whether the user has accidentally scraped parts of the anatomy adjacent to the "target".

As the demands on medical practitioners increase, the pressure is increasing to find effective alternatives to traditional teaching practices that reduce reliance on teaching provided by expert mentors. VR medical simulation is being demonstrated as an important component of teaching reforms. Most importantly, medical simulation protects patients by allowing training to take place before contact with real patients. Simulation also provides opportunities for new types of learning not normally possible in many medical scenarios. For example, trainee surgeons can practice any number of times on simulated patients without requiring access to cadavers or necessarily even their mentor. These advantages, combined with a full control of the simulated environment, and the ability for the training experience to be refined and developed year after year, make simulation a critical component of medical training that will grow and mature as the potential of new technologies are realised.

2.1 Learning Modalities

Learning can be grouped into three basic learning modalities: auditory, visual, and kinaesthetic. It has been shown that an individual will typically learn well in one particular modality and will learn less effectively when information is presented in either of the other two modalities [6, 46]. This section explores how virtual reality simulation caters to each of the learning modalities and hence provides potentially better learning than traditional teaching.

Visual learning is well catered for by virtual reality. Today's computer graphics are approaching photo realism (though this isn't necessarily the optimal visual style at all times). VR environments provide a unique opportunity to depict environments and structures in ways that are impossible using other media. Key structures can be highlighted, inanimate structures can be animated, and tools can be manipulated using pre-recorded animations. All of these freedoms give VR the potential to create more effective training experiences.

A good mentor will verbally explain what they are doing and why. Hence, auditory learners are well catered to by the expert mentor in the traditional master-apprentice approach currently commonly used in surgical training. Although simulations have a long way to go before natural language interaction with an

expert system is possible, it is relatively easy to employ text-to-speech or pre-recorded phrases to enrich the learning experience offered in VR.

It could be argued that kinaesthetic learning can be provided using the common user interface of the mouse [161]. However, fully realising the potential of kinaesthetic learning requires specialised computer-human interfaces such as tactile displays [116, 158] and haptic devices [63, 127, 140]. These interface devices no longer operate solely to input data to the computer, but also deliver tactile and haptic experiences to the user. This is particularly relevant to medical part-task trainers such as intubation simulations, tonsillectomy simulations, and dental simulations [148], where the key to proficiency is learning how to manipulate tools and structures effectively. Although the fidelity of current haptic interfaces may be questionable, there is clearly potential for haptically enabled VR medical simulation to enable trainees to learn how tissues feel and more generally learn kinaesthetically.

Recent research by Ferguson *et al*. [48] suggests that "students with a 'convergers' learning style tend to perform better" leaving students with the hands-on "accommodators" learning style less well catered to. Accommodators prefer hands-on experience as a way of learning. Haptic interaction caters to this type of learner in new ways and thus provides new opportunities to improve learning outcomes.

Desktop haptic devices are particularly promising in delivery of kinaesthetic learning experiences where the range of motion typically used in the real procedure can fit within the workspace of the haptic stylus. In such situations it is possible to replicate the procedure's workspace at the same scale as in reality. This scenario can be further enriched by aligning the visualisation with the stylus workspace thus minimising the leap-of-faith some other simulations may require.

In summary, VR simulations can cater well to each of the learning modalities. Of particular interest are recent advances in computer-human interaction that provide new ways to interact, such as haptic devices. These advances create new (simulation-based) opportunities for kinaesthetic learning that is common in medical undergraduates [12], especially within surgical disciplines.

2.2 VR Medical Simulations: The State of the Art

VR simulation has come a long way in the past decade, and has now reached a point where it has been demonstrated to effectively improve learning outcomes in clinical settings [134]. This section details a selection of the best and most relevant VR medical simulations currently available.

2.2.1 Dental and Bone Surgery Simulators

Although bone surgery and dentistry differ in terms of the anatomy and in the wider context, the critical interaction for both is the manipulation and remodelling of rigid structures (bones or teeth) during surgery. Hence, the technology developed for these simulators is very similar. These simulators commonly employ volumetric models of rigid structures, here referred to as voxel-based surgical simulation.

Voxel-based surgical simulation technology has much in common with medical image processing. Medical imaging data are often used in surgical planning [132]. These are the same data as used by voxel-based simulations. There is a lot of overlap between the two representations (medical imaging for surgical planning and voxel-based datasets for simulation). The techniques used to visualize computed tomography (CT) or magnetic resonance images (MRI) are increasingly being employed in computer graphics [33] and certain types of medical simulations, the most significant of which will be summarised here.

Researchers at Stanford University have produced several haptically interactive VR medical training simulations including Temporal Bone Surgery [97], Craniofacial Surgery [98], and Dental Surgery [148]. The type of interaction supported by each of the simulators is very similar: volumetric tissue removal (or re-positioning) of rigid bone (or tooth) tissue with haptic feedback. The haptic feedback is computed in real time from the interaction of the tool with the voxel-based representation of the bone or tooth. Further, additional fidelity and effect is added to simulate the forces resulting from the motion of the abrading tool tip. However, this approach does not support volumetric tissue removal from soft or deforming tissues.

Other bone surgery simulators have also been developed. Most notably, Voxel-Man TempoSurg is a commercially available temporal bone surgery simulator developed by the Voxel-Man Group with the University Medical Center Hamburg-Eppendorf, Germany. TempoSurg includes support for the import of patient-specific data from CT scan data.

In 2007, Tolsdorff *et al*. used TempoSurg to "to evaluate the quality of patient-specific models as well as the benefit of preoperative simulation for the surgical procedure to follow" [152]. In the 20 cases involved in the study, they found that the quality of preoperative rehearsals of middle-ear surgery was substantially improved. Moreover, concerning the adequacy of the models, Tolsdorff *et al*. write, "the quality of simulation [was] close to exercising with cadaveric specimens with the decisive advantage that it does reflect the patient's individual anatomy". This demonstrates that certain types of VR medical simulators are "coming of age" where realism and learning benefits are clear. The use of voxel-based techniques as applied to tissue simulation will be discussed in more detail in Chapter 6.

2.2.2 Intubation Simulators

Endotracheal intubation is a difficult and risky procedure that is commonly performed on patients in order to maintain a clear airway and for administering a general anaesthetic [60]. The procedure can be briefly summarised as using a laryngoscope to manipulate the tongue and insert a tube into the trachea.

The interaction between the laryngoscope and the tongue is key to the success of the simulator. The tongue must deform realistically in real time in response to contact with the laryngoscope. Realistic visco-elastic modelling of the deforming tongue is especially challenging. The tongue is a large muscle with internal variability and dynamic behaviour even when the patient is unconscious, which makes it particularly difficult to simulate well.

Rodrigues *et al*. were the first to create a real time interactive mechanical model of the tongue [121-123]. Their biomechanical model of the upper airway included all key elements at, by today's standards, relatively low resolution. It included interconnected mechanical models of "the tongue, ligaments, larynx,

vocal cords, [and] bony landmarks" [122]. The finite-element-method (FEM) was employed to simulate the behaviour of the tongue, "from simple linear elastic material to complex non-linear visco-elastic material" [122]. Resolution of the models was quite low to run on the available hardware (for example, the laryngoscope consisted of 35 thin-shell elements). The described configuration of the mechanical system is intricate; it includes specialised hidden mechanical interconnections to impart the desired properties [121]. This suggests that the system is highly specialised and used some innovative techniques to enhance the mechanical characteristics of the system. Empirical validation of the model was performed which showed "that the non-linear model behaves most closely to the experimental studies" [123]. The visual quality of their simulator is poor by today's standards; models consist of low numbers of flat-shaded polygons with no texture images, bump-maps or the like to add realistic surface detail.

In 2003 Mayrose *et al*. reported on their development of a virtual reality intubation simulation [87]. Models were derived from the North American National Institute of Health (NIH) Visible Human dataset. They employed a mass-spring based mechanical model to provide real time haptic interaction with volumetric 3D models. Visual realism was not given high importance and consequently the simulation's visual realism is low. No validation of the mechanical model or the simulation as a whole was performed. Chapter 9 provides a more detailed description of intubation simulation generally and also describes a new intubation simulation (ISim) that I have developed using the tissue simulation framework described in Chapters 4 to 8.

2.2.3 Eye Surgery Simulators

Since Sinclair and others developed the first ophthalmic simulators over a decade ago [137], a number of eye surgery simulations have been developed ([65, 74, 78]). Eye surgery simulators are one application where real time soft-tissue is simulated, but where the simplicity of the morphology has made it possible to produce useful simulations before other more complex scenarios are possible. Of particular interest is Faure *et al*'s Ophthalmic Surgery Simulator, which simulates the vitrectomy procedure (removal of part or all of the clear gel known as the "vitreous

humor" located between the lens and the retina), and was built using the Simulation Open Framework Architecture (SOFA) Framework [4]. Although the simulation has not been clinically evaluated, the available documentation describes simulation elements such as haptically interactive cut-able membranes. It is clear from the published work that the simulation is reasonably realistic and is capable of delivering a high level of realism to users. However, despite the use of SOFA, the development of support for the key interactions was clearly quite complex. More information about SOFA is provided in Chapter 3.

2.2.4 Minimally Invasive Surgery and Endoscopic Simulators

Minimally invasive surgical (MIS) procedures (also known as keyhole surgery) such as laparoscopy, and arthroscopy are well suited to VR simulation largely because the user interface is relatively easy to replace with equivalent devices interfaced to a computer. When performing a real MIS procedure the operator is guided visual feedback from an optic-fibre camera displayed on a monitor, and haptic feedback via the handpieces of the surgical instruments. Since visual feedback is delivered from a monitor, not directly from the anatomy of the patient, exceptional simulation validity can be achieved. Likewise, endoscopic procedures use visual feedback via the endoscope (presented on a monitor) and haptic feedback via the endoscope handle.

Although creation of realistic interfaces for MIS or endoscopic simulations is simpler than other types of VR medical simulations, these simulations must address some unique challenges based around the fact that the key interactions take place within tightly enclosed spaces. Handling collisions at haptic refresh-rates under these conditions is particularly challenging. Consequently, much of the research conducted into developing MIS simulators has focussed on this area. This technology is important to advancing VR medical training simulations beyond the level that is currently possible.

Commercial simulators for a range of medical procedures that use easily mimicked user interfaces (such as MIS, arthroscopy, and catheter insertion) have been developed by a number of companies. These simulators are the most refined type of VR simulators available due largely to the close match between the

simulation user interface and the interface of the real medical devices. Hence the user interaction with the simulator very closely approximates reality. The most significant products have been developed by Simbionix USA Corporation, Mentice AB (Sweden) (which acquired Xitact, Switzerland), Immersion Corporation (US), Medical Simulation Corp (US), Voxel-Man Group (Germany), VirtaMed AG (Switzerland), SimSurgery AG (Norway), and Surgical Science AB (Sweden).

Aside from the simulators produced commercially, several MIS and endoscopy-type simulators have recently been developed by research groups. Hellier *et al*. [64] at the Australian Commonwealth Scientific and Industrial Research Organization (CSIRO) have developed a multi-threaded colonoscopy simulation framework. The simulator provides robust simulation of the deformation of the colon, high visual fidelity, and a specially built haptic interface [128] that enables the user to manipulate a real endoscope with tactile feedback. Their work provides an excellent solution for "cavity simulators". However, surgical simulators and VR medical simulations more generally cannot be developed using this framework without additional technology.

Several arthroscopy simulators have been developed [51, 166], although recent activity developing this specific application appears to be limited [88]. These simulators are the least relevant to this thesis because they typically make minimal use of deformable structures and all interactions are via specialised (keyhole) instruments, whereas the ideal tissue simulation technology would be more flexible and applicable to many different medical simulation types.

Harders *et al*. at ETH Zurich (Swiss Federal Institute of Technology) have developed an impressive hysteroscopy simulator [61, 62] that is now being commercialised by VirtaMed AG (and distributed in partnership with Simbionix). This simulator renders the uterine cavity with high visual fidelity and allows the user to cut deformable polyps and myomas. Ablation is also supported. According to their paper [62], their approach to cutting is "optimized for our specific application domain" and is relatively complex, requiring further work before arbitrary cut paths in 3D are supported.

In summary, the number of VR medical simulations is growing. Whilst graphical realism is high, realistic haptic feedback and interactive tissues remain a

key challenge. Without better core technology all simulation development must overcome similar obstacles. High quality medical simulation-based training requires new technology to enable realistic interaction. As a field, medical simulation research will repeatedly need to develop similar capabilities until these capabilities become readily available in shared development tools.

Chapter 3. Simulator Development Tools

Software libraries provide reusable components to reduce the programming effort required to implement a given task. Many libraries exist that can simplify development of medical simulations. Partly due to the demand of computer entertainment, libraries and development tools that give developers the ability to deliver realistic interactive computer graphics are quite common. However, surgical simulations rely on subtle tactile and visual cues that are difficult or impossible with existing libraries, although there are smaller libraries which can be useful in contributing to a solution.

Selecting and combining libraries is a good way to avoid re-implementing common tasks. Care must be taken to ensure that libraries, written with a certain usage in mind, do not prevent implementation of key features. Hence, despite the prevalence of game development application programming interfaces (APIs) and open-world simulation APIs (commonly used for flight or combat simulations), there are very few software libraries that cater directly to medical simulation developers.

The technologies to use to develop a VR medical simulation must be very carefully selected. Allegiance to a particular approach to delivering critical design objectives (such as support for key interactions) will determine the algorithms, and in turn the APIs used. Libraries can substantially reduce the time and effort required to develop a simulation. Thus, before designing a simulator it is useful to review the existing libraries and consider which components can be utilised. Good technology selection and software design will result in a minimal amount of development effort to unify the available software libraries and implement new features to create the finished application. The more features a software library contributes to the features required, the more enticing it is to incorporate the library

into the final design. However, care must be taken to ensure that the interfaces required are available and that the libraries chosen don't force developers to corrupt the design in order to conform to inconvenient architectures. This section reviews available software libraries which provide important functionality and discusses their relevance and significance to VR medical simulation development.

3.1 Software Tools

Virtual reality medical simulations must render 3D scenes effectively. Typically scenes contain a number of models where each object must be rendered with different surface properties and lighting effects. This requires management of resources such as textures, shaders and other assets such as normal-maps. Although it is possible to manage these assets and tasks manually, there are a number of tools which can reduce the development effort needed and provide efficient methods of managing rendering the 3D scene.

3.1.1 Scene Graphs

A scene graph is a tree or graph data-structure that stores a set of assets used to render a scene (models, textures, shader programs). The use of a scene graph is essential for scenes consisting of large numbers of objects, especially when only a small fraction of objects from the entire scene are visible at any given time. Scene graphs are designed to optimise rendering operations by employing fast sorting and searching algorithms to perform tasks such as occlusion culling, z-sorting, and batched render calls to render scenes more efficiently. There are a number of scene-graph application programming interfaces (APIs) available, each with different feature sets and nuances. Scene-graph APIs are useful in developing any virtual-reality application. However, care must be taken to select an API that provides maximum utility without inhibiting implementation of key features such as interactive tissue simulation.

Medical simulations, and surgical simulations in particular, have virtual scenes different to most other types of simulations and visualisations where scene graphs are typically employed. Surgical simulations typically consist of enclosed environments densely populated with interconnected structures. Under such

circumstances there may be little benefit, if any, in performing per-object occlusion tests or scene-graph sorts. Instead, a per-polygon occlusion sort is required since deformable objects will occlude parts of other objects depending on current deformation and positioning (which rarely occurs in more open environments). Hence, many of the advantages typically provided by scene graph APIs do not apply to the types of scenes commonly found in surgical simulations. However, some features still provide significant advantages in this context.

3.1.1.1 Asset Loading and Run-time Data Management

One important advantage of some scene-graph APIs is the simplified loading of assets such as models, textures, and shader-programs and their storage in a manner optimised for rendering. Without a ready solution, implementing these features can be time consuming. There is not only the problem of loading the assets, but also managing them at run time. There are also a number of complicating factors such as storing textures in suitable formats that are compatible with the graphics hardware that will run the simulation and the shader programs. At some stage these problems must be addressed by the developer. Some scene-graph APIs allow the developer to remain ignorant of many of these details.

3.1.1.2 Rendering and Automatic Shader Resource Bindings

Shader fragment programs (shaders) define how transforms and lighting will be applied in order to render a given lighting effect (refer to section 3.1.4, page 23). Some scene-graph APIs support automatic shader bindings. This feature is especially useful because it removes the need for the developer to explicitly manage binding of textures, matrix transforms, and other fragment program dependencies between the application and the shader program. Moreover, shaders can define rendering algorithms that require several passes. Each shader pass can be thought of as a new overlay to the rendering effect (though multi-pass shaders need not always operate in this manner). Without automatic shader binding, multi-pass shaders require specialised run-time code that must match the definitions in the shader. Changes to the shaders will require changes to the run-time code and vice versa. This can be time consuming for developers to maintain and is another potential source of coding errors.

3.1.1.3 Disadvantages

Scene-graph APIs can reduce flexibility and introduce unnecessary complexity at various levels. For example, low-level variables may be locked by the scene graph API (this is a common way for scene graphs to prevent changes to the scene graph before sorts are performed to accelerate rendering) or even completely hidden from the developer. Having access to low-level data such as vertex buffers is especially important when working with new graphics-processor APIs that use graphics memory for both graphics and more general purpose computations. Avoiding unnecessary copies of this data substantially improves performance (discussed in more detail in section 3.3). Many scene graphs do not expose the vertex buffer resource identifiers to the developer; instead these are managed by the scene graph itself.

Ogre 3D is an active open-source scene-graph project with numerous commercial games and 3D applications to its credit [72]. It is relatively well documented and well supported by its online developer community. It is cross-platform and uses an abstraction layer that allows it to render using DirectX 9 (Microsoft's graphics API) or OpenGL 2 (Open Graphics Library). There are many plugins available to extend functionality. Of particular interest is the "oFusion" scene importer which includes an exporter for exporting complete scenes from popular 3D modelling software such as Autodesk's 3D Studio Max™. Automatic shader binding is provided by the closed-source run-time library. Unfortunately the closed-source run time prevents access to low-level data structures (vertex buffers etc) needed for efficient use with parallel programming APIs such as Nvidia's Compute Unified Device Architecture (CUDA). Ogre 3D is available under the Lesser GPL license, which is quite permissive.

Scenix (formerly NVSG) is available for free from Nvidia. Source code is available by special arrangement to owners of some high-end products or by purchasing a developer license from Nvidia. The license agreement does not require the payment of any royalties or imply the sharing of intellectual property rights of applications that use it.

Scenix manages scene loading and rendering. The asset pipeline used to prepare models and scenes is relatively simple and quite powerful. Scenes prepared

in 3D Studio Max can be exported (as ".3ds" files) into Nvidia's FXComposer which then provides tools to apply and edit shaders and then export the complete scene. The scene is then ready for run-time use via the Scenix run time. All asset loading and shader bindings are handled by the API. This provides excellent flexibility and enables the use of any shader program (multi-pass, post-render effects and so on) with minimal development effort.

Scenix is also cross-platform. Written for use in C/C++, it makes extensive use of templates to enforce correct use of data types. For example, attempts to write data to variables of read-only types results in compilation errors rather than run-time errors. This provides developers with more immediate feedback and enforces cleaner, more explicit code.

Scenix is strongly object-oriented and allows developers to use inheritance to create custom data types. This can be used to neatly integrate specialised features such as the tissue simulation.

3.1.1.4 Other Scene Graphs

Numerous other excellent scene graphs are available. A more detailed review of these APIs is beyond the scope of this thesis. Links to the leading APIs are provided for completeness below.

OpenSG (http://opensg.vrsource.org)

Open Inventor (http://oss.sgi.com/projects/inventor)

PLIB (http://plib.sf.net)

SGL (http://sgl.sf.net)

OpenRM (http://openrm.sf.net)

Open Scene Graph (http://www.openscenegraph.org)

Performer (http://www.sgi.com/products/performer)

3.1.2 Game and Simulation Engines

Game and simulation engines (that are typically commercial) provide excellent support for high-quality real time rendering of interactive virtual environments.

However, they are targeted at interactive entertainment (mostly computer games) development and do not support key interactive elements required in medical simulations such as interactive tissue models. However, they set the standard for graphical realism and to a lesser extent animation. As tissue simulation and other key technologies for medical simulation are developed, these engines may integrate the features required to make them a leading option for accelerating medical simulation development. In the meantime we can learn from them and monitor their evolution to ensure that opportunities to leverage this technology are not missed.

For the past several years, Crytek GmbH have set the standard for graphics realism in computer entertainment. Crytek released the game Far Cry in 2003. The game was built using their game engine, CryEngine. CryEngine was outstanding in its completeness and use of cutting-edge rendering techniques such as High Dynamic Range (HDR) lighting effects, realistic shadows, and fluid effects, although the fluid effects are limited to oceans, pools and puddles and their interaction with personnel and vehicles. These effects have much in common with the fluid effects required for medical simulations, but are not sufficiently similar that the engine would be suitable; pools of saliva and blood would not be well represented by effects that model water, and techniques used to model surface disturbances caused by personnel and vehicles are not capable of realistically modelling fluid behaviour in surgical simulations.

The CryEngine engine throttles well; rendering effects are automatically scaled depending on the capabilities of the available hardware. Crytek have continued to release outstanding game engines, namely CryEngine2 and CryEngine3. However, when considering development of medical simulations, these engines serve as a benchmark as to what is possible rather than having any direct application for a number of reasons, in particular: license costs are high, medical simulations are rarely open-world environments, and the engine does not support haptic interaction or tissue simulation.

There are a number of other game and open-world simulation engines available. These engines may be useful for emergency medical simulations and other larger-scale (open world) simulations. The technology employed is very

capable with respect to rendering realism. As commercial opportunities in medical simulation increase, these engines may be adapted for use in medical simulations provided key capabilities are supported directly or effectively integrated.

3.1.3 Leading Commercial Game and Simulation Engines

CryEngine, by Crytek GmbH (www.crytek.com)

Gamebryo, by Emergent (www.emergent.net)

Id Tech 5, by Id Software (http://www.idsoftware.com/business/idtech5/)

Microsoft ESP, by Microsoft (www.microsoft.com/ESP/)

Torque 3D, by Garage Games (www.garagegames.com)

Unity, by Unity Technologies (www.unity3d.com)

Unreal Engine, by Unreal Technology (www.unrealtechnology.com)

The Unreal Development Kit is now available for free use.

Marks *et al*. have recently published a more detailed review of Game Engines for use in medical simulation development [85]. This work supports my opinion that whilst game engines can accelerate and simplify the development of some types of VR medical simulations, they do not provide support for soft-tissue models or haptic interaction which is central to the successful development of the majority of VR medical training simulations.

3.1.4 Rendering APIs (OpenGL and DirectX)

Graphics hardware technology has advanced very rapidly in the past ten years, and particularly in the past five. To provide access to growing flexibility and programmability, rendering APIs also have evolved. During this time, hardware has moved from being capable of only fixed-function pipeline rendering, to versatile, fully programmable shading, and finally to general purpose computing.

Software access to graphics hardware is typically provided to application developers via one of two APIs: Microsoft's DirectX, and Silicon Graphics' OpenGL (Open Graphics Library). Historically OpenGL has set the standard. More recently OpenGL has failed to fulfil its potential as DirectX introduces new

capabilities. DirectX has pushed forward the boundaries by adding new hardware-accelerated pipeline stages and support several unique features. Which API to use will become a more significant decision as feature sets diverge.

OpenGL is a cross-platform open standard for real time computer graphics. Graphics device manufacturers implement their device drivers to conform to the specification. Open-source implementations of OpenGL such as Mesa 3D are also available. Open-source drivers allow developers to optimise execution on custom hardware such as Sony, Toshiba and IBM's (STI's) Cell Broadband Engine (CellBE), and they enhance low-level algorithms for new capabilities. As alternative approaches to real time rendering such as micro-poly, micro-voxel and real-time ray tracing are explored, the prevalence and significance of alternative rendering engines will likely increase.

Since the release of DirectX 10 the differences between the two APIs has grown. As of DirectX 11, Microsoft have introduced new shader stages for programmable hardware-accelerated tessellation, hull shading, sub-division patches, and Bezier patches, none of which have equivalents within the OpenGL specification at the time this was written. These new features provide opportunities for new methods of efficient rendering of many millions of polygons per frame. The limits of the traditional rendering pipeline consisting of transform, cull, rasterise, and lighting are being approached as the average polygon size becomes smaller than a pixel.

The recent improvements to DirectX came too late to impact the choice of technologies for implementing the tissue simulation described in this dissertation. In 2007, Nvidia's CUDA API was at version 1.0. It had full OpenGL support and limited support for DirectX (for example, geometry could not be rendered from DirectX from CUDA memory without being copied) [108]. Consequently, OpenGL was used with CUDA to develop this tissue simulation.

3.1.5 Physics APIs

The key interactions in surgical simulation involve real-time modelling of interactive deformable structures. Capturing the behaviour of interacting rigid bodies and articulated bodies (for example tubes) is also required to improve

realism and increase immersion. Physics APIs provide these features via simplified interfaces, and hence are a useful tool for enhancing virtual reality simulations with interactive physics-based effects with minimal development effort.

PhysX was originally developed by Ageia as the first physics API optimised to run on specialized parallel-processing hardware referred to as the PhysX Processor. In 2006, with the release of CUDA, a new version of PhysX that executes on the GPU was released. PhysX is based on technology developed by Novodex AG. Today, it is one of the top physics engines used in interactive entertainment. PhysX is now owned and maintained by Nvidia Corporation.

PhysX is capable of handling large numbers of rigid bodies and articulated bodies. Moreover, it supports real-time interactive cloth simulation with variable simulation characteristics. Most importantly (for surgical simulation) it is capable of simulating fluids and deformable soft bodies. All features can interact with each other and the user, though in testing this can be critically limited for certain types of interactions important to medical simulation.

In PhysX soft bodies can be simulated using either volumetric meshes, or shell meshes. These volumetric meshes are based on tetrahedra. This type of soft-body simulation exhibits more realistic mechanical characteristics compared to shell meshes. However, the structure of the volumetric-mesh is fixed; it cannot be cut like the shell mesh. Shell-mesh based soft bodies can use an internal gas pressure constraint to improve the behaviour. Without this constraint the mechanical behaviour is significantly less realistic if used to simulate living tissue. However, the gas pressure constraint cannot be used where the object may be cut or ablated (as is often required in surgical simulation).

PhysX supports collision detection and handling of the entire scene. Unfortunately, experiments with PhysX show that collisions between soft-bodies or cloth with other soft-bodies or rigid-bodies are not detected sufficiently reliably for haptic interaction. Even with careful tuning of vertex-spacing in colliding object pairs, objects may pop through one another.

Since neither approach to soft-body simulation supports key interactions, PhysX has limited utility in medical simulations. PhysX does however have much

to offer in managing secondary objects in the simulation. For example, PhysX can simulate the surgical cloth draped over the patient, or the behaviour of tubes or wires connected to instruments. These are not of critical importance though they may improve immersion. The question for developers remains: Does PhysX provide enough useful functionality to justify the time required to integrate it into the simulation? The answer depends entirely on the application.

Support in other physics APIs for GPU accelerated calculations is growing; a critical feature if cloth or deforming soft bodies are important. Havok Physics not far behind PhysX. The open source project Open Dynamics Engine (ODE) has seen declining support and limited growth over recent years. Conversely, Bullet Physics is an open-source project that is growing rapidly and already includes GPU accelerated features.

OpenTissue is a collection of works (many by K. Erleben) maintained by the Datalogisk Institut på Københavns Universitet (DIKU) (Department of Computer Science at the University of Copenhagen). The collection is quite diverse and includes some interesting works on elastically deformable solids, fluid simulation, and collision detection. The source code of demonstration applications is provided, although documentation can at times be sparse. These works may be useful stepping stones, although reviewing what exactly is on offer can be time consuming.

In summary, there are a number of physics APIs with growing feature sets that typically cater to the requirements of interactive entertainment and open-world simulations. Depending on the functionality sought, these APIs can provide developers with ready access to implementations of optimised physically based simulation algorithms. Unfortunately, none of these APIs are targeted directly at medical simulation development. This is particularly evident in soft-body intersection handling, which is either not supported or is insufficiently reliable for haptic interactions that are central to VR surgical simulations. Hence, the use of physics APIs in medical simulation is limited to providing supporting effects and capabilities rather than the support for the core interactions.

3.1.6 Collision APIs

Collision detection and computation of the collision response are amongst the most processor-intensive tasks that must be handled in real-time VR simulations. Algorithms, techniques, and libraries have been developed to reduce the amount of processing required while maximising the reliability of detection and realism of the collision response. This section discusses collision detection for surgical simulation and reviews the leading techniques and libraries.

Collision detection is the process of identify intersecting objects, and often the geometric primitives (tetrahedra, triangles, edges and lines or points) that intersect. Details of precisely which primitives are colliding are commonly required to compute an appropriate collision response such as the rebound trajectory of a bouncing ball.

Once collisions are detected, a collision response must be computed. Details of the intersecting primitives and other data (e.g. the location of the collision relative to the centre of mass, friction models etc) are used to un-intersect the models without causing artefacts (e.g. jitter, popping etc), deform and deflect surfaces, and exchange kinetic energy between colliding objects.

3.1.6.1 Collision Detection for Surgical Simulation

Collision detection in surgical simulations is particularly challenging because models are typically deformable and densely spaced. Deforming objects can fold, and folds can result in self collisions, which are contacts between different parts of the same model. The number and complexity of objects in a given volume of surgical scenes gives rise to more collisions than open virtual worlds. Hence, not all algorithms employed in more sparse scenes are suitable.

Collision detection is typically performed in two stages: a broad-phase pass identifies intersecting volumes potentially containing intersecting objects for the second pass, and a narrow-phase pass that identifies individual primitives that are intersecting.

Developers of VR medical simulations can choose to develop collision-detection systems from scratch, they can use the collision API from a physics API

or, they can use a special-purpose collision API. This section summarises the most significant collision APIs currently available.

Collision detection and response is a pre-requisite of most physics simulation capabilities. However, even though Physics APIs include these capabilities, the software interfaces to enable collision detection to be performed by the developer are often simplified and do not provide access to the types of data required to support, for example, tactile feedback. Therefore, physics APIs like PhysX, Havok, and ODE, while containing excellent collision detection and collision response capabilities, do not provide access to the low level algorithms surgical simulation developers require.

OPCODE is a small collision-detection library developed by Terdiman in 2001 [144]. The library uses a bounding volume hierarchy based on an axis-aligned bounding-box (AABB) tree. OPCODE is optimised for minimal memory usage. It is capable of fast detection of triangle-triangle collisions on conventional hardware. However, OPCODE is not optimised for fast updates to the AABB tree. Nor is it designed for use on parallel hardware such as GPUs. This limits its attractiveness for use in medical simulations since any data structure employed for accelerating collision detection must be capable of efficient updates to mesh changes because meshes in medical simulations are typically non-rigid.

The SOLID collision-detection library was originally developed by van den Bergen in 2001 [156]. It is uses an iterative algorithm for computing the distance between objects that was originally described by Gilbert, Johnson and Keerthi in 1988 [55, 154]. SOLID includes optimisations for fast updates to deformable solids [153]. It also supports fast penetration depth estimation [155] which is essential for haptic interaction and computation of reactive forces.

SOLID is optimised for execution on the CPU but it does not natively work using polygonal meshes. Instead, objects are internally represented as primitive shapes and complexes of polytopes. To overcome this limitation the author suggests that another library (Qhull [7]) be used to decompose complex objects into a compatible format. Clearly, while SOLID has promising functionality well suited to application in medical simulations, it is not ideal.

RAPID is a research project developed at the University of North Carolina based on oriented bounding box trees [56]. It recursively subdivides polygonal objects and groups the subdivisions for fast collision tests. One criticism raised by Terdiman is the relatively large memory footprint when compared to OPCODE [145]. Terdiman addresses these deficiencies by re-engineering RAPID into another library named Z-Collide [145]. Z-Collide removes some redundant data stored in tree nodes and replaces the matrices used to represent rotations (3 x 3 = 9 elements) with normalised quaternions (4 elements). Neither RAPID nor Z-Collide are optimised for deforming objects. Finally, V-Collide "combines I-COLLIDE's sweep 'n' prune with RAPID" [67].

H-Collide is a more recent collision detection library optimised for use in haptic applications [58]. It uses a hybrid hierarchical representation that uses spatial partitioning to separate objects occupying large areas into a hash-table for efficient lookups, an oriented bounding box tree (OBB-Tree) within hashed volumes to accelerate lookups of small sets of potentially intersecting primitives, and frame-frame coherence to accelerate lookups based on previous results (since changes between frames are minimal) [58]. H-Collide has been used to produce haptically interactive applications based on rigid objects. However, like most of the existing libraries, it is not optimised for deforming structures where the additional processing overhead of updating the collision detection system at run time is not considered.

Whether it is beneficial to leverage existing libraries must be decided early in the development of any simulation. Even if the key interaction of surgical simulation is not possible with existing technology, there are numerous tools available that provide efficient solutions to some of the issues a simulation developer must address. Physics and collision detection APIs provide highly optimised and efficient software components that provide robust solutions for adding physics-based animation to simulations at different scales. Scene graph APIs streamline the process of scene creation and greatly simplify the complexity of run-time components required to render realistic graphics effects in real time. With careful consideration and design, combining these components carefully will produce higher quality simulations with far less developer effort.

3.1.7 Medical Imaging Tools

Medical simulations require anatomically accurate 3D models. These models can be created by 3D artists. However, the time and skill required to create sufficiently detailed and realistic models can be reduced by using medical scan datasets to create templates, or ideally ready-to-use models derived entirely from real patient data. Depending on the type of VR application, these models can be unique to a given patient or completely generic. If the process used to create sufficiently accurate 3D models is reproducible and not dependent on large amounts of user input or artistic input, then new types of patient-specific applications are possible. Unfortunately automatic generation of simulator content directly from patient data is not yet widespread.

Reviewing these tools in more detail is beyond the scope of this thesis. The most significant of these are listed in Table 1 (below). Common capabilities include the ability to segment volumetric data and generate shell mesh 3D models.

Table 1: Leading Medical Imaging Tools

Name	Company Name or Open Source
VTK/ITK	Open source
3D Slicer	Open source
Amira	Visage Imaging GmbH
BioImageXD	Open source
caBig-XIP	Open source
CT-Analyser	Sky Scan
Farsight Toolkit	Open source
MedINRIA	Open source
Osirix Viewer	Open source
ParaView	Open source
Scan IP	Simpleware
True Life Anatomy	True Life Anatomy Pty Ltd
VisTrails	Open source
VolView	Kitware Inc.
VR-Render	IRCAD

3.1.8 Simulation APIs

An insight into the complexity of developing a VR medical simulation can be obtained by considering the size of the leading game engine development teams: For example, Id Software (id Tech 3) employs around 115 employees [2], Crytek (CryEngine 3) more than 500 employees [34], and Epic (Unreal Engine) approximately 140 employees including 21 full-time programmers [80]. Fortunately, the development of VR medical simulations can be simplified. However, whatever simplifications are employed, there is no way to avoid the additional complexities created when supporting real time tissue simulation and haptic interaction. Fortunately, simulation APIs can reduce the development effort required to develop medical simulations. However, significant challenges remain for VR medical simulation developers irrespective of the APIs employed.

Clearly the development of medical VR simulators is complex and combines numerous technologies and techniques. This section provides an overview of the most significant simulation APIs available for VR medical simulation development. These APIs combine some or all of the previously described APIs into a single, unified API.

3.1.8.1 SOFA: Simulation Open Framework Architecture

The Simulation Open Framework Architecture (SOFA) is an open-source project founded by researchers of the Alcove group at INRIA [31]. The project aims to provide an architecture to facilitate development of simulators using a modular architecture that maximises component re-use while "minimizing the impact of this flexibility on the computation overhead" [31]. It is being actively maintained and developed by developers at INRIA and more widely contributed to by researchers such as the CIMIT Sim Group, ETH Zurich, and CSIRO [139].

The current version of SOFA (1.0 beta 4) [139] includes support for different types of deformable models based on mass-springs or linear and co-rotational finite element method (FEM) (discussed in Chapter 5), fluid models, and

a number of collision detection and response methods. Each component is configurable via XML to facilitate experimentation and simulation development.

SOFA is designed for use in medical simulation development. A scene graph representation is used to describe not only the rendered scene but also the simulation components that combine to produce interactive simulation elements such as interactive tissues. The component types include mechanical system models, surface representations, collision detection algorithms, and constraint solvers. As such, SOFA is maturing to become an invaluable contribution to the field of medical simulation research. However, the use of such a versatile architecture is not without its limitations.

Flexibility in software architectures is a trade-off with specialisation. Specialisation brings speed and higher performance. It therefore becomes a question of whether sufficient performance can be achieved while retaining the desired flexibility. With the rapid changes occurring in general purpose GPU (GPGPU) APIs (refer to section 3.3.3) the overhead of re-engineering flexible interfaces is particularly challenging. For example, CUDA has evolved from C-only language support to include C++ language features. Retaining flexibility of components written for radically changing APIs is difficult and time consuming. Hence, although SOFA has a lot to offer, it requires developers to conform or adapt to its architecture.

GiPSi (General Physical Simulation Interface) integrates open source libraries (including TAO, OPCODE, and OpenHaptics) to simplify development of re-usable organ-level simulation components for tactile medical simulation [24]. Unfortunately the project no longer appears to be active (the last release of the API was 29-Oct 2008), and it does not include any re-usable components for tissue simulation.

Other simulation APIs include NeuroVR [43], SPRING, VRASS, SSTML, and ISReal [45]. None includes support for capabilities targeted at medical simulation development, such as deformable models, and therefore are not considered relevant to this discussion.

3.1.9 Debriefing and Assessment APIs

VR medical simulation-based assessment should never completely replace the assessment performed by expert practitioners. However, there are a number of ways that VR medical simulations can improve the quality, depth and breadth of assessment. Further, debriefing is recognised as an invaluable method for improving learning outcomes [130], [44]. Simulation offers new mechanisms for data collection for more comprehensive debriefing.

3.1.9.1 Improving and Streamlining Debriefing with VR Simulation

Development of expert skill requires deliberate practice with informative feedback [41]. Feedback and reflection are important learning tools that are underused in medical education [19]. Debriefing provides a forum for trainees to receive feedback and reflect on their performance during simulation-based training. Commonly audio and video recordings of the students' performance during the simulation session are used during debriefing. VR simulation can go further by allowing all interactions, as well as the complete state of the simulation itself, to be recorded. This removes any potential for problems related to insufficient video camera vantage points where it may not be possible to see, in sufficient detail, critical aspects of the interaction. It also creates new opportunities for recording simulation sessions without requiring specially outfitted rooms so that many students can be simultaneously recorded in non-synchronised sessions cheaply and simply directly via the simulation software. These recordings can then be batched and expertly reviewed. Batching review is more efficient for reviewers than participating in whole sessions. This increases the depth and quality of feedback and improves standardisation since performances can be reviewed literally side-by-side. Also, since the interaction of the student with the simulation can easily be reviewed without identifying the trainee, feedback can be more objective and completely free of any bias.

Capturing interactions via VR simulators provides opportunities for developers to streamline assessment in new ways completely independently of whether the assessment itself is at all automated. VR simulation software can be used to identify the user's progress during a training scenario. Moreover, this can

be extended to automatically record time markers where the trainee performs milestones of a procedure. Assessors and reviewers can thereby skip to important parts of the session. Further, if replayable session data is collected, rather than video from a number of virtual cameras, the reviewer can manipulate the virtual camera perspective during replays to see, and highlight, issues with the student's performance that may have been hidden behind obstructing structures for the student. These alternate vantage points can be provided as part of the delivery of feedback as small videos to enhance the value of the feedback.

3.1.9.2 Scoring

There is research that suggests simulation-based assessment is better than written examination [96]. However, "Simulation-based training provides minimal feedback and relies heavily on self-assessment." [83]. In a 2006 review of the accuracy of self-assessment the majority of studies "demonstrated little, no, or an inverse relationship" between self assessment and external assessment [35]. Clearly there is scope for improvement. VR simulation in particular offers unique opportunities for improved assessment methods. By utilizing the facility of VR simulation to capture all user interactions it should be possible to develop algorithms and systems to process user interactions to automate key parts of assessment and thereby achieve better validity and standardisation of assessment.

Development of meaningful scoring algorithms is a challenging area of research that currently remains in its infancy. There are a number of approaches worthy of further investigation including; motion analysis [167], dexterity measures, collision penalties, time and event based penalties. No assessment or automatic scoring APIs were found at the time this was written.

As the fidelity of simulators improves and more specific skills are targeted it will be increasingly important to have verified performance metrics. Looking beyond definition of meaningful performance metrics, standardisation has the potential to solidify the credibility of simulation-based assessment. Ultimately, VR medical simulation offers new solutions to be used by certification and regulatory bodies to certify competence.

3.1.10 Conclusions

Surgical simulation and medical part-task or procedural trainers that rely on haptically enabled interactions with soft-tissue can, to some extent, be developed with existing tools. However, the perfect tool for developing VR medical simulations does not exist. There are many that contribute different features and capabilities, but none is ideal. SOFA is an excellent starting point. However, in order to produce a flexible and efficient solution, a range of techniques and technologies must be combined. Doing so efficiently is especially challenging particularly when supporting haptic interaction.

With the right combination of tools, simulation development has the potential to become accessible to a wider audience. Use of these tools will enable researchers and developers alike to focus their efforts to make increasingly more specialised contributions, thus enabling rapid improvement in the quality of VR medical simulators.

3.2 Literature Surveys

When required features are not available as part of existing software libraries it is necessary to develop these capabilities by building upon existing libraries and techniques. Recent surveys by Nealen *et al*. [105], Teschner *et al*. [147], and Klein *et al*. [75] describe the basic models, concepts, and algorithms relevant to interactive VR surgical simulator development. The literature as it relates specifically to each of the system components is provided at the beginning of subsequent chapters.

3.3 Recent Advances in Parallel Computing Hardware

“Computer science is no more about computers than astronomy is about telescopes”, said Edsger Dijkstra in 1967. Just as an astronomer benefits from a detailed understanding of telescopes and the principles and manner in which they operate, in order to develop good software, a computer scientist must first develop an understanding of the principles and nuances of the processing hardware.

3.3.1 The Importance of Knowing Your Hardware

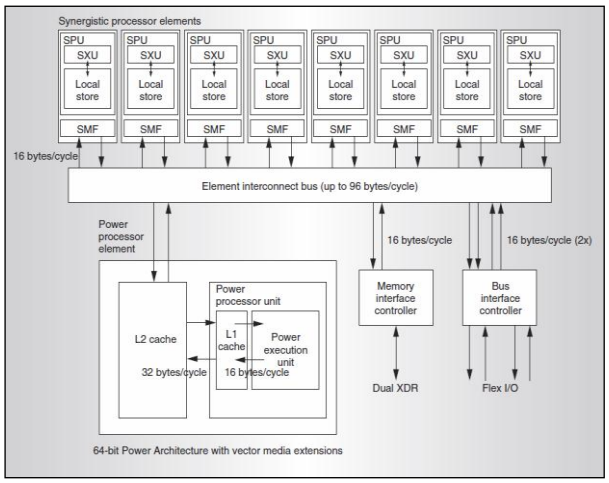
Programming interactive virtual reality simulations demands efficient utilisation of hardware resources. It is simple enough to display a rendered 3D model, or even an animation. However, to make this animation sufficiently realistically interactive to teach surgical skills to a trainee surgeon requires numerous systems to work together efficiently in hardware and software.

When an algorithm must run at a very high rate, specialised hardware can be developed. Taken to the extreme, a fully programmable gate array (FPGA) could be used to run complex algorithms in very few clock cycles. Conversely, high-performance general-purpose chips can be used to emulate more specialised (and expensive) hardware completely in software. There is thus equivalence between hardware and software. An algorithm can be implemented as specialised logic with specialised hardware or as a program that can be executed on more general hardware. The more closely that the software "fits" the hardware that it executes on, the more likely it is going to be able to fully tap the hardware's processing capacity and achieve peak performance. However, some compute devices require a more abstract approach. For example, Sony-Toshiba-IBM's (STI) Cell Broadband Engine Architecture (CBEA) uses complex caching to avoid stalls of several hundred instructions, which can occur in the event of a cache miss [66] although as Breitbart *et al*. point out, by using software caching "the developer loses the chance to utilize double buffering, which is one of the most important benefits of the CBEA" [20]. It is therefore important to understand the nuances of the hardware for which real time software is developed so that good performance can be attained.

3.3.2 The Cell Broadband Engine Architecture

With the release of the latest generation of computer gaming consoles the CBEA went mainstream in Sony's Playstation 3. This processor's unique architecture differs significantly from graphics processors. It has however been demonstrated to be capable of handling single instruction multiple data (SIMD), and also more inherently serial algorithms, substantially faster than CPUs [27] and in some cases

as fast as GPUs [91, 92][1]. CBEA compares favourably with GPUs, particularly when considering performance versus power consumption [27].



Each Cell consists of a 64bit PowerPC core and 8 SPEs. Each SPE has 256KB of memory, a memory controller and a SPU with an SIMD unit and 128 16B registers. The main bus has an internal bandwidth of over 300 GB/s for transfers between SPEs. [14]

Figure 1: The Cell Broadband Engine architecture [59]

CBEA favours throughput over latency. Latency can be hidden using prefetch (double buffering) and caching. Since CBEA exhibits strength in a diverse range of applications [27] it is worth considering its use for tissue simulation. It then becomes a question of cost versus gain. As pointed out by Buttari *et al*. [23] CBEA requires a deep understanding of the architecture in order to develop applications that take full advantage of it. Although this is typical for programming specialised high-performance software generally, it is my impression that the complexity of developing software for the CBEA significantly exceeds that of the alternatives. Perhaps as tools such as RapidMind's dynamic compiler [89] mature, the complexities of developing high-performance software for CBEA may be substantially reduced. However, in their current state, CBEA development tools and compilers do not sufficiently simplify the task of developing software for this platform. Consequently, it is not utilized by the systems described in this thesis.

[1] Barry Minor (Senior Technical Staff, IBM) is quoted as having said on his blog, "we found that using only 7 SPEs for rendering a 3.2 GHz Cell chip could out run an Nvidia 7800 GT OC card at this task by about 30%". This blog is no longer available, though its content is quoted in forums (see reference in main text).

3.3.3 General Purpose Graphics Processing Units

This section will introduce General Purpose Graphics Processing Units (GPGPU) with a brief history of 3D computer graphics, because computer graphics for the entertainment industry has driven the evolution of programmable graphics hardware to a point that has resulted in a revolution for high-performance computing on readily available parallel hardware (GPUs).

3.3.3.1 A Brief History: From Early Computer Graphics to General Purpose Parallel Computing

When considering the best approach to a software design problem, it is useful to consider how the hardware that executes it is changing. There is no point investing in software that relies on specialised compute capabilities that are likely to soon become obsolete.

In the 1980s, the first 3D applications were beginning to appear and as markets for computer games grew, the demand for better 3D computer graphics grew also. Computer games have been a driving force in the development of 3D computer graphics technology, and since the first 3D games appeared there has been healthy investment in new software and hardware to deliver better 3D computer graphics.

The first 3D computer games weren't all truly 3D. Sometimes referred to as 2.5D, games such as Wolfenstein 3D were limited to vertical and horizontal edges only and small colour palettes. This was necessary in order to achieve real time performance on the available hardware of the day. Clearly in this era, the CPU's compute power was a limiting factor preventing better 3D rendering.

Later, once markets had grown, the impetus for more powerful graphics-oriented hardware drove the development of hardware designed specifically to accelerate graphics computations. Early graphics accelerators used a fixed-function graphics pipeline to rasterize, transform, and light 3D geometry and render 3D scenes. Vertex-processing units (vertex "shaders") and pixel-processing units (pixel shaders) were initially implemented as separate specialised processors. This made processing difficult to balance between the two shader types, leaving a significant proportion of processing units waiting idle while the other did its

processing. This problem was addressed with the introduction of unified shaders that supported a larger instruction set in order to perform the dual functionality of vertex processing or pixel shading. Hence all unified shaders contributed to processing most of the time, regardless of the relative amount of vertex or pixel processing required.

GPUs have since then not only grown in raw processing power to support rendering increasingly complex scenes, larger textures, and output at higher resolutions (Figure 2), they have also become more flexible. Where once developers could only program the way in which multiple textures were combined to achieve effects such as transparency, today's hardware has several programmable pipeline stages that are far more flexible and have allowed new tasks to be performed on graphics processors.

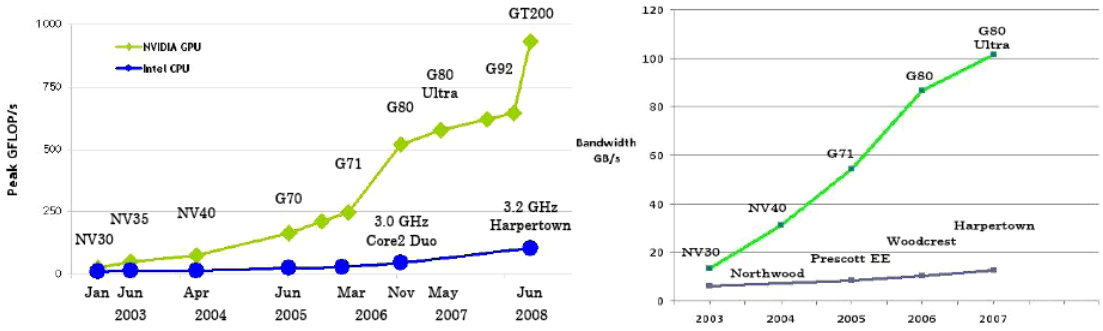


Figure 2: GPU and CPU processing power (left) and memory bandwidth (right) [108]

By encoding data into textures, developers have used the graphics processor to accelerate numerous tasks including matrix algebra and physics computations [49]. This approach adapts the programmable graphics pipeline to a wide variety of new tasks. However, it requires a deep understanding of graphics concepts and adaption of a system to new tasks for which it was not originally intended. New APIs such as Brook/Brook++ [22] have greatly simplified developing general purpose software that utilises the parallel processing power of the GPU. However, this approach is no longer necessary with the release of general purpose computing APIs that execute natively on new GPGPU hardware.

Developer support for GPGPU software development is improving; for example, the Nsight GPU debugger has recently been released. Moreover, GPGPU hardware continues to rapidly grow in power and reduce in cost. Clearly GPGPUs

are here to stay and any software developed for GPGPU is unlikely to quickly become irrelevant.

The following section details the hardware architecture of a recent GPGPU that has been designed to be a flexible massively parallel processor.

3.3.3.2 The Nvidia GT200 Hardware Architecture

The GT200 hardware architecture is Nvidia corporation's second GPGPU hardware series. The first was the G80 series which was released in 2006 headed by the GeForce 8800. Due for release mid 2010 is the recently announced Fermi architecture, which looks set to continue the trend of advancing processing power, memory bandwidth, and programmability. The Fermi architecture will also introduce a number of features targeted at scientific computing, server applications and general purpose high-performance computing including ECC error correction and native double precision floating-point operations. However, a detailed review of the Fermi architecture is beyond the scope of this document.

Nvidia's GeForce GTX280 is the flagship product of the GT200 series range. Peak performance is an impressive 933 GFLOPs for single-precision floating point computations [135] (some 20 times the peak performance of a Intel's recent CPU, the 3.3GHz W5590 Intel Xeon [69]). Clearly the GT200 architecture has enormous potential, but potential is nought unless it is attained. Let us now explore the philosophy of GPGPU, then the specifics of this architecture before discussing how best to utilise this new architecture and whether it is even suitable for tissue simulation at all (and the constraints its use imposes).

"The GPU is specialized for compute-intensive, highly parallel computation – exactly what graphics rendering is about – and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control" [108]

The GPU is a massively parallel processor designed, like the CBEA, for throughput rather than minimal latency. Rather than using approaches such as branch prediction and out-of-order execution to accelerate thread execution (as in modern CPUs [136]), GPUs have many simpler cores with minimal caching.

When effectively programmed, latencies are concealed and the massive compute power of the device can be fully utilised.

3.3.4 The GT 200 architecture

GPUs manage tens of thousands of threads with effectively no thread management overheads [126]. Specifically, the GeForce GTX280 has 240 streaming processors (SPs) grouped into 30 streaming multiprocessors (SMs). This allows the GTX280 to execute up to 30,720 concurrent threads (128 per SP) [54]. Each SM has 16KB of shared memory. Each group of 3 SMs has a 24KB L1 Texture Cache and 64KB of constant memory. Also each SP has its 2KB share of the 64KB SM register, and each SP can access the 256KB L2 cache or global memory (up to 4GB). The architecture is illustrated in Figure 3.

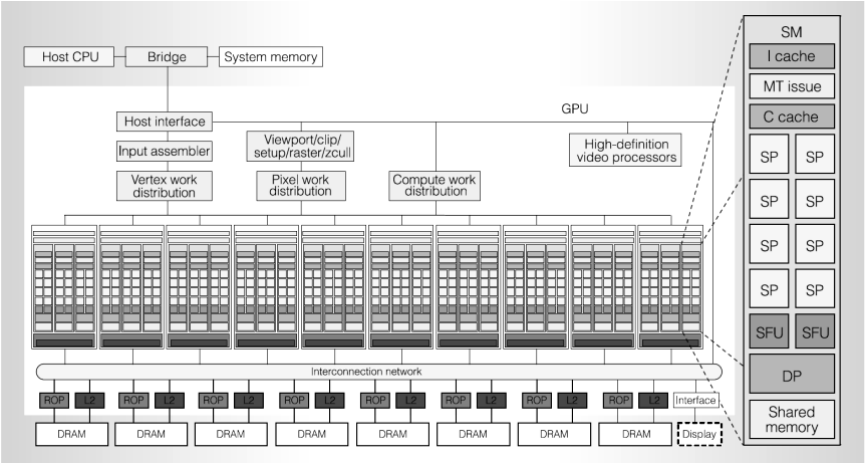


Figure 3: The hardware architecture of the Nvidia GT200 GPU [54]

GPUs excel at data-parallel processing [54]. The ideal algorithm will be arithmetically intensive with minimal data dependencies, and be readily parallelizable. Alternatively, GPUs can be used for more general, mathematically expressed formulations to problems. Tissue simulation can be approached a number of ways, some more suited to the GPU than others. The following sections discuss how to best simulate tissue on the GPU, and how to develop code for the GPU and the APIs involved.

3.3.5 Introduction to CUDA

With the growing processing power and expanding feature set of the graphics hardware, efficient ways of programming for the GPU have grown in complexity. Until recently, this had been solely focussed on improving graphical realism of rendering using more advanced lighting, texture filtering, and surface modifier effects like bump mapping and parallax mapping. The application programming interfaces (APIs) giving access to the features of the GPU provided programmable shaders, then later new programmable processing stages were added (vertex shaders and geometry shader stages), which gave developers more freedom to modify geometry and explore a whole range of new techniques using the processing resources of the GPU. These new stages, with their SIMD capabilities, were optimised for vector operations. Soon after, it became increasingly common to use the GPU for many tasks other than computer graphics.

When CUDA was released in 2006 it was a significant step forward. Although other APIs (for example Stanford University's Brook [22] and AMD/ATI's Close-to-Metal) provided abstraction layers to simplify development of general purpose software for the GPU, CUDA was the first to provide a C-like interface to the resources of the GPU and overcome many of the limitations of existing GPGPU APIs. For example, CUDA provides multiple output stream scatters, access to high-speed local caches, single native instruction synchronisation, and buffer bindings with graphics APIs to reduce the need for multiple redundant copies of data.

CUDA was developed by Nvidia and is currently only compatible with Nvidia graphics processors. Since its release, numerous CUDA applications have been developed which attain over 99% of the theoretical peak processing performance, including molecular dynamics simulation [142], and MRI processing [126]. It is CUDA's ability to provide access to all levels of memory in a relatively straight-forward manner that sets it apart from other GPGPU APIs.

3.3.6 Other GPU Programming APIs

GPU programming has recently been greatly simplified for more general purpose programming (no longer just rendering). New APIs provide high-level

functionality while retaining low-level access to the compute resources of graphics processors as general purpose massively multi-threaded processors. Aside from CUDA, there are two other APIs that are gaining support, features, and capabilities; Khronos Group's OpenCL [103] and Microsoft's Compute Shaders [18].

Stanford University's Folding@Home [77] project was the first mainstream GPGPU application. The API used by the project is Brook/Brook++ [22], which works only with ATI/AMD graphics processors. ATI/AMD have also released Close-to-Metal, which provides GPGPU programmability for their graphics hardware.

Shader Model 5 will be included within DirectX 11. The latest DirectX SDK from Microsoft contains a tech preview which uses an emulation layer to run the code examples on the CPU. DirectX 11 also includes new programmable graphics pipeline stages including tesselator, hull shader (NURBs: Non-uniform Rational B-Splines), and domain shader (patches). Usage of pipeline is relatively fixed, but stages may still be useful since they are highly optimized.

OpenCL is an open-standard GPGPU language specification. Implementations have been released to run with Mac OS and other operating systems and graphics cards. OpenCL differs from other GPGPU APIs as its scope includes support for transparent parallelization. Parallel code can run on any available processor (single core, multi-core, or GPU) that supports OpenCL.

3.4 Conclusion

VR medical simulations consist entirely as a combination of hardware and software. The software is composed of numerous components that must operate in efficient unison in order to achieve a reliable and compelling user experience and hence achieve the best possible learning outcome. The amount of development effort required depends largely on how successfully existing software libraries can be reused. These libraries must be selected carefully in order to ensure that their promise can be attained in the context of the simulation software application. While new libraries are being created all the time, and existing libraries are maturing, new advances in computing hardware provides new opportunities to

perform aspects of processing that previously limited the quality and realism of components key to medical simulation. Hence it is particularly important that any libraries used do not inhibit efficient utilization of these new hardware. Although many useful libraries aim to reduce the development effort required to develop VR medical simulations, features that implement critical aspects are still evolving rapidly to take advantage of recent advances in computing hardware. Interactive tissue simulation that delivers reliable and compelling user interaction is one such component that requires further development in order for VR medical simulations to advance to the next level.

Chapter 4. A New Tissue Simulation Framework

Many medical interventions involve cutting or removing tissue. These interactions rely on visual and tactile feedback to inform the practitioner in a number of ways including whether more pressure is needed to cut to the required depth, or whether hidden structures are present. The simulation of these tissues is especially challenging to perform in real time because of the processing required, and the complexity of delivering tactile feedback from a changing model.

Stable mechanical simulation of the behaviour of soft tissues typically requires considerable processing power. A common strategy for reducing the processing required is to use pre-processing to simplify parts of the algorithm used to perform real time updates. However, this complicates systems where user interactions alter the structure of the mechanical simulation system. This thesis introduces a new tissue simulation framework (TSF) that combines relatively simple components to deliver a versatile and compelling interactive tissue simulation that can readily be modified and maintained in real time.

4.1 The Tissue Simulation Framework

The TSF maximises the detail of the visible model by separating the rendered mesh geometry from the mechanical simulation sub-system. This separation allows the framework to reduce the processing requirements of the mechanical simulation without reducing visible detail. It also simplifies model preparation and widens the possibilities for implementing each sub-system since each representation of the model is independent of the other. Moreover, the separation of mechanical and visual components allowed each of the sub-systems to be built and tested in

isolation, which greatly simplified development, trouble-shooting, and optimisation.

Soft tissues encountered during surgery exhibit diverse and complex characteristics such as non-linear visco-elastic deformation. This is compounded by variations within the tissue, and the interconnections between tissues that are commonly found throughout the body. Rather than focus on simulating the mechanical dynamics of these tissues with absolute accuracy, the TSF employs approaches that target simplicity and efficiency. This allows higher-resolution models to be used, which in turn increases opportunities for the system to model a broader range of tissues. Additionally, by employing multiple instances of the system, resolution can be added only where it is needed.

Simplicity is important for a number of reasons. User interactions with the system (such as cutting and ablation) will cause structural changes that the system must handle reliably and accurately. The methods used to model the tissues' mechanical behaviour are designed to handle these changes efficiently, and without adding additional complexity or processing load to the system.

The structures found in living systems are amongst the most elaborate and complex. The structures of the sinus cavity, for example, are both intricate and convoluted (see Chapter 9 for an example of the TSF being implemented for sinus surgery simulation). The TSF reduces the workload of modelling these structures by utilising commonly used volumetric medical scan data as the basis of the 3-dimensional models. Optionally, this data can also be used to vary the mechanical properties of the tissue volumetrically, which means that a single instance of the TSF can exhibit varying mechanical properties that correlate with the volumetric densities from the scan data. The separated high-resolution visible model is easily generated directly from medical volumetric scan data (resolutions of 256 cubed voxels, approximately 17 million voxels) as demonstrated in Chapter 6. By circumventing the need to manually model complex anatomical structures, the TSF reduces the time and development effort required to use it in a diverse range of applications.

Numerous medical interventions and part-tasks rely on tactile feedback to inform user interactions. Delivering a compelling tactile illusion is particularly

challenging because our sense of touch is sensitive to higher frequency variations than our vision. Visual updates at 60Hz are enough to deliver the illusion of smooth and continuous video, but tactile rendering requires updates of the order of 1kHz to deliver a smooth and compelling tactile experience. This requirement imposes significant new challenges on the realisation of the tissue simulation because the mechanical simulation, already a potential processing bottleneck, must model deformations at far higher rates to support reliable haptic rendering.

The importance of high performance and efficiency are two-fold. Although simulating the mechanical dynamics of a changing model at interactive rates is our primary concern, optimal performance also increases the level of detail of the model that can be achieved at interactive rates. The separate mechanical simulation and visible model allow the fidelity and resolution of the mechanical dynamics to be traded with visual fidelity and realism. High-resolution representations of both sub-systems would be ideal and, as hardware processing capacity continues to increase, the TSF will remain relevant by allowing the use of increasingly complex models.

To sustain processing performance required for real time response, the TSF utilises the processing capabilities available on the graphics processing unit (GPU). The evolution of the traditional fixed-function graphics pipeline into a fully programmable pipeline has allowed more general computations to take advantage of the processing available on the GPU [111].

However, processing on the GPU is typically only beneficial on Single-Program Multiple-Data (SPMD) structures. As such, determining and integrating aspects of the algorithms suitable for processing on the GPU is crucial to ensure beneficial gains in simulation performance. Furthermore, the synchronisation of data between the GPU and the central processing unit (CPU) remains necessary as the haptic device (controller for user interactions) is managed by the CPU.

In summary, by separating the mechanical simulation system from the high-resolution volumetric model the tissue simulation can be adapted to a wider range of applications. Scenarios requiring mechanical simulation of large volumes can use a larger mechanical simulation node size to efficiently span the model

without requiring excessive processing. Conversely, if fine cuts are needed then the volumetric model resolution can be tuned to more closely match the mechanical simulation so that any alterations to the visible model are accurately reflected in the mechanical simulation.

4.2 System Overview

The remainder of this chapter introduces the complete TSF. The system is described in more detail in subsequent chapters.

The TSF comprises four representations of the 3-dimensional tissue model (Figure 4, below). A high-resolution volumetric model is used to enable simple and robust interactive cutting and tissue removal. This volumetric model is used to generate a shell mesh (rigid render mesh model) that is optimised for efficient, high quality rendering using conventional lighting and shading algorithms. A coarse mechanical simulation that matches the topology of the other models is used to deform the render model. The memory required to store the multiple representations is negligible compared to the processing and bandwidth savings it achieves.

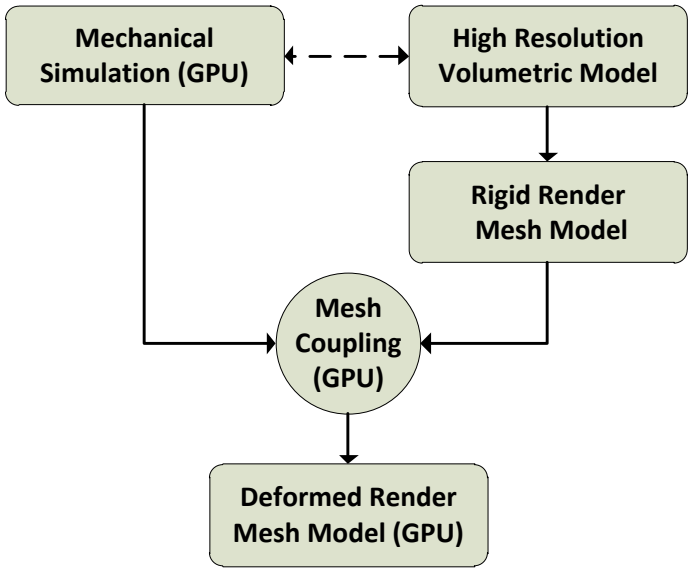


Figure 4: Tissue Simulation Framework block diagram

The high-resolution volumetric model simplifies cutting and ablation by allowing fast, simple collision detection. When a cutting instrument intersects occupied voxels of the volumetric model, those voxels are deleted. Only the changed sub-

volumes are then evaluated to update the closed surface that forms the rigid render mesh model (Figure 5). This output render mesh is optimised for efficient high-quality rendering using common shading and lighting algorithms.

The mechanical simulation is updated at high rates using the GPU. High update rates are important for two reasons; 1. The mechanical behaviour of the system will remain stable for a wider range of material characteristics when the elapsed time between updates is small (Chapter 5). 2. A high update rate is needed to achieve high quality haptic rendering (Chapter 8).

Changes made by the user to the high-resolution volumetric model update both the rigid model and the mechanical simulation. Updates to the render mesh are direct and implicit via the algorithms employed to create the render model. In order to maintain the mechanical model as an accurate analogue of the render mesh, changes to the volumetric model are used to update the coarse volumetric mesh used by the mechanical simulation. This is required to ensure that the representative models remain coherent, which ensures that the TSF behaves realistically when, for example, dissected tissue is moved away from the substrate.

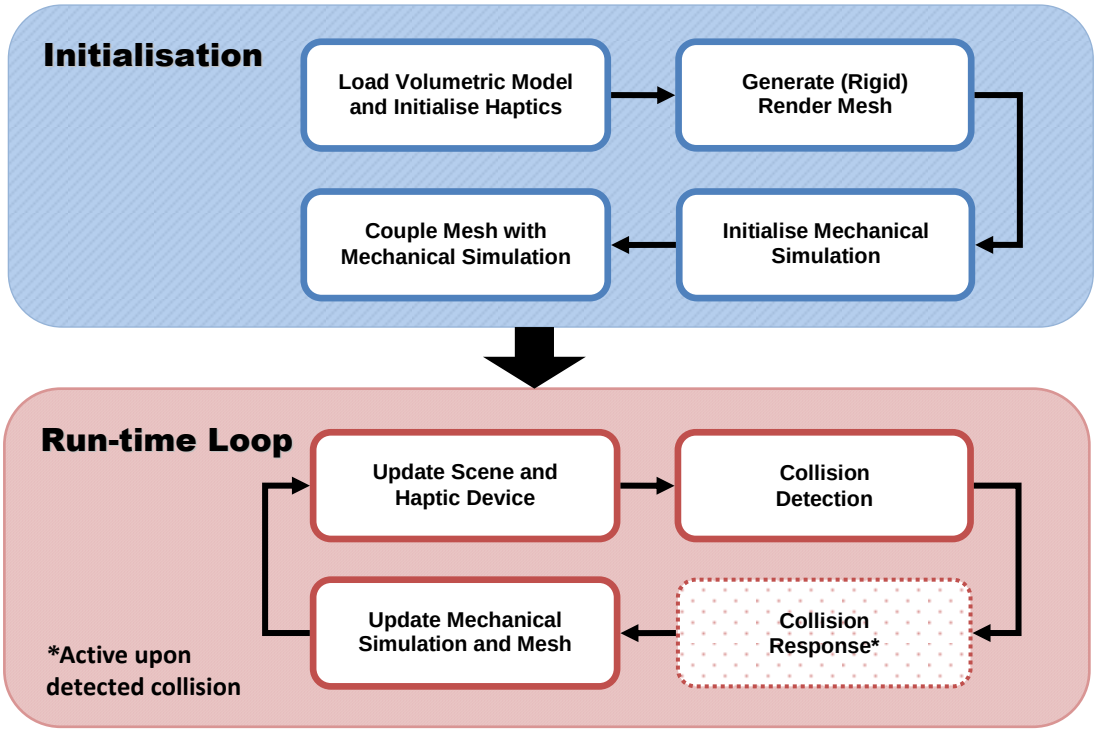


Figure 5: Initialisation and run-time loop

The interactively cut-able and ablatable high-resolution model and the state of the mechanical simulation must be combined to produce the interactive soft tissue. Avoiding excessive memory transactions between the CPU and GPU is imperative to ensure that the overall system performance is not constrained to the lesser performance of this memory interconnect. The render mesh geometry must be copied to GPU memory for rendering to screen. Hence, the TSF uses the GPU to map the deformed state of the mechanical simulation onto the interactive high-resolution render mesh to produce a deformed render mesh model. The deformed render mesh model is then rendered directly from the GPU without the need for additional memory transactions between CPU and GPU. Furthermore, the mesh coupling processing is performed at the lesser rate of 60Hz since its output is required solely for generating the final render (refer to Chapter 8 for details of how haptic-rate updates are avoided in this part of the system).

Collision detection has the potential to consume large amounts of processing resources. The TSF avoids this by re-using fixed (constant) mappings between sub-volumes of each representation of the model. Contact determination is reduced to simple spatial mappings that require negligible processing resources. These efficiencies are used to achieve haptic rendering at the required update rate. Since the TSF uses multiple representations of the same model, haptic rendering algorithms were developed that explore the efficacy of the different options. They are described and compared in Chapter 8.

The TSF achieves its aims (detailed in Chapter 1) by avoiding common pitfalls of existing techniques. Chapter 9 presents a number of examples of how the framework has been used to add critical functionality to a range of VR medical simulations that would not be possible with existing development tools (Chapter 3). The algorithms developed for each component build on the strengths of existing tools (Chapter 3) and techniques to provide new functionality in a reusable framework.

Chapter 5. Mechanical Simulation

This chapter describes the mechanical simulation component introduced in Chapter 4. The design targets efficiency and versatility ahead of realism to create a system that is capable of modelling a diverse range of mechanical behaviours and in so doing can be applied to the simulation of a range of tissue types commonly required in medical simulations.

The mechanical simulation component is the software system used to simulate the physical behaviour of the tissue in response to user interactions (Aim 3) and different mechanical loads in real time. Real-time mechanical simulation is notoriously processor intensive and the subject of continued research as applications demand higher fidelity and resolution at interactive rates. VR medical simulation training imposes additional requirements, such as cut-ability and higher update rates for haptic rendering, on the tissue simulation. These additional requirements reduce the number of approaches to mechanical simulation that can be used.

Mechanical simulations typically model objects as numerous interconnected sub-elements. The configuration and type of sub-elements, together with the algorithms employed to model their inter-relationships, define the material dynamics. Typically, the mathematical expression that defines the relationship between sub-elements is homogeneous and must be evaluated for each update. These types of mechanical simulations are ideal candidates for parallel execution.

There is good potential for fast execution of certain types of mechanical simulations on GPUs because of the repetition of computations across a large number of independent nodes. However, there are a number of challenges to achieving new levels of flexibility, topological complexity, and plausible

mechanical behaviour, particularly at the rates required for haptic interaction. The type of sub-elements used and the material dynamics model must be carefully designed in order to execute efficiently while providing the additional features demanded by VR medical training simulation. For example cutting and tissue removal, when implemented using the finite-element-method (FEM), "are non-trivial and require comprehensive book-keeping and computer power to work in practice" [21].

This chapter describes a novel approach to efficient and versatile mechanical simulation that is engineered to efficiently leverage the massive available compute power of GPUs whilst adding new capabilities essential to the success of VR medical training simulations.

5.1 Background

The two most significant approaches to mechanical simulation are the Finite Element Method, and the Damped Mass-Spring model. Each method has its own nuances and can be implemented in numerous ways to provide different capabilities. One key challenge is adapting them to support cutting and ablation to enable their use in real time VR medical simulations.

Underlying these two approaches to mechanical simulation is a 3-dimensional lattice of interconnected nodes or elements. Different lattice arrangements can be used to produce changes in the mechanical characteristics of the simulation. Moreover, different structural arrangements result in different mechanical attributes and affect the overall complexity and stability of the system. More importantly, the lattice arrangement also impacts the ease with which support for cutting and other interactions can be incorporated into the system.

5.1.1 Structure and Valence

Most approaches to real time mechanical simulation of deformable structures employ a static representation of 3-dimensional structure. This structure not only defines the locations at which material behaviour is modelled, but also the interconnections through which changes to the model are propagated. The number of interconnections within the structure determine the valence. Specifically, the

valence is the number of connections of a given node to neighbouring nodes. Different structures or arrangements of connections will result in different valence. When the mechanical system itself uses a higher-order representation such as the finite element method (see section 5.1.2.1) minimal valence is all that is required to have a stable mechanical simulation. However, mass-spring based approaches to mechanical simulation (see section 5.1.2.4) typically require higher valence in order to avoid problems such as folding or popping (where nodes jump unpredictably to achieve lower entropy) or to produce specific mechanical properties. A minimally valent approach described by Teschner et al. [146] avoids popping and folding by utilizing a volume preservation constraint in the computation of internal forces.

5.1.1.1 Tetrahedral Volumetric Meshes

In 3-dimensions, the minimum valence required in order to fully constrain a node is four. This results in a tetrahedral mesh structure which can take various forms. A regular tetrahedral mesh is composed of uniform tetrahedra. This limits the mechanical model's ability to represent the rendered model directly (since regular tetrahedra cannot form any arbitrary 3-dimensional shape). However, regular tetrahedral grids provide a number of advantages. The regular structure has known interconnections, which simplifies lookups of neighbours without the need for stored adjacency data. Also, regular grids have uniform density and, when used with mass-spring systems, minimise problems with folding while retaining minimal valence.

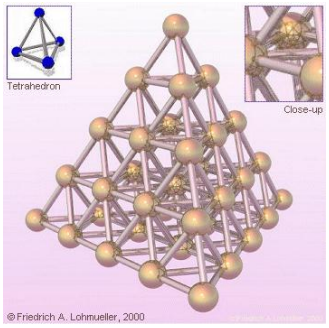


Figure 6: A regular tetrahedral mesh [81]

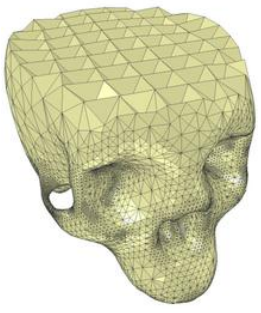


Figure 7: An irregular tetrahedral mesh [94]

Irregular tetrahedral meshes can vary greatly. Poorly formed tetrahedral meshes, just like poorly formed 2-dimensional triangular meshes, can include slivers

(tetrahedra with greatly varying edge lengths) or very large or very small tetrahedra. At best a mesh with highly variable size or shape will merely waste computations. However, irregular tetrahedral meshes can remove the need for multiple representations of the same object at run time by using the same lattice for modelling mechanical behaviour and rendering. Moreover, if a system can support mesh irregularities then it may simplify addition of support for cutting and other effects.

Finally, other advantages of tetrahedral grids include simplified barycentric coordinates (coordinates centred about a tetrahedron's centroid) that are useful for binding other datasets to the deformed coordinate space of the mechanical simulation. For example, texture information (volumetric or surface textures) and higher-resolution shell-meshes (for accurate rendering) can be bound to the mechanical simulation to create visually detailed deforming models without the high computational cost associated with simulating the mechanics directly.

Tetrahedral meshes are non-trivial to generate. However, there are tools available to generate them. For example, TetGen, PhysX, and ADINA.

5.1.1.2 Cubic Volumetric Meshes

Unlike tetrahedral meshes, cubic meshes allow for very simple implicit addressing, which can facilitate more efficient implementation of the simulation system by reducing the computations and memory usage normally associated with looking up the location of connected nodes. This is particularly important for GPU-based implementations which are often memory bandwidth limited. Cubic lattices also allow the developer to map node data directly into cubic texture memory, which can be significantly faster than more abstract approaches where direct use of texture memory is not a viable option.

Common cubic lattice arrangements have direct equivalents in 2-dimensions. The simplest lattice connects nodes only to their nearest neighbours orthogonally (Figure 8). In 3-dimensions this results in a valence of 6 (for non-edge nodes). Without additional constraints, this arrangement is highly susceptible to folding and is inherently unstable since nodes are not sufficiently constrained. Diagonal springs, often referred to as "shear-springs", can be added to stabilise the

mesh (Figure 9). Further mechanical stability can be achieved by extending the connectivity beyond the nearest neighbour. Figure 10 illustrates this concept in two dimensions by adding secondary connectivity. Many arrangements are possible by extending the range of connections even further and optionally interconnecting any or all possible pairs of nodes.

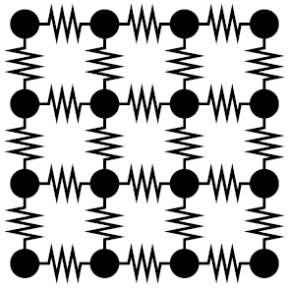


Figure 8: Simple square-lattice [129]

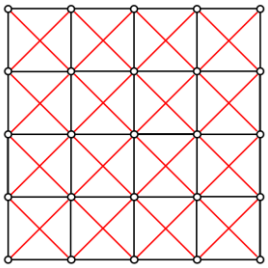


Figure 9: Square lattice with diagonals [138]

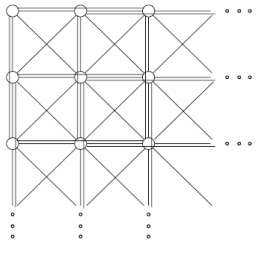


Figure 10: Square lattice with diagonals and secondary connectivity [118]

5.1.1.3 Adaptive and Hybrid Schemes

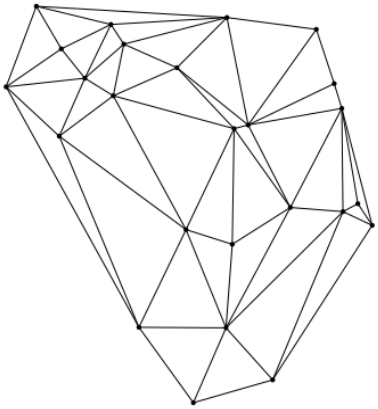


Figure 11: A 2D Delaunay triangulation [68]

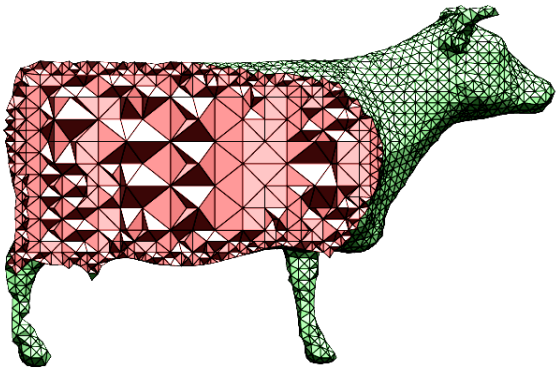


Figure 12: Adaptive tetrahedral tessellation [76]

When a single mesh is to be used for mechanical simulation and rendering, or where additional accuracy is required around areas of higher detail, adaptive tessellation can be used to create the volumetric mesh. A common problem with adaptive mesh generation is large variations of the angles between element edges (dihedral angles). Large dihedral angle variations are indicative of an irregular mesh () which in turn will reduce the fidelity of the mechanical simulation unless special provisions in the simulation are made to normalise the stiffness of springs based on their arrangement. Stiffening the spring connecting nodes on the shortest

edge will improve the triangle's (or tetrahedron's) tendency to be less resistant to compression and consequently potential problems with "popping".

Adaptive tessellation techniques can also be employed to handle changes to a mesh during cutting.

5.1.2 Real-time Mechanical Simulation Techniques

Real time physics modelling of deformation has its roots in mechanical engineering where non-real time simulation has long been used to evaluate and analyse the mechanical characteristics of everything from components and containers to architectural designs. The methods used are generally mathematically well defined and understood. However, a mechanical engineer need typically only know if a certain load will cause a component to fail whereas a tissue simulation for VR surgical simulation must model a dynamic system with changing structure. Some approaches have been adapted to such usage, while others prove difficult despite considerable research effort having been expended.

5.1.2.1 The Finite Element Method

The Finite Element Method (FEM) represents an object volumetrically as a finite set of elements, usually tetrahedra. FEM-based mechanical simulations use a number of simplifying assumptions to find approximate solutions, via numeric integration, to a set of partial differential equations (PDEs) that model each element's structural mechanics. Considerable research has been performed into adapting the FEM to real time applications. Different methods have been found for improving efficiency, though achieving sufficient efficiency for real time use while adding support for surgical interaction remains a challenge. This section summarises significant works based on FEM relevant to the VR medical simulation applications.

Müller and Gross [102] describe a real time FEM formulation that supports "elasticity, plasticity, melting and fracture". This work demonstrates excellent stability of the mechanical simulation, particularly under large rotational deformations. Its ability to simulate fractures suggests that it may be possible to adapt their approach to surgical applications.

Morris [101] describes how to create models with calibrated mechanical properties based on FEM for real time usage. Similarly, Sedef *et al*. [131] describe how to perform "Real time Finite-Element Simulation of Linear Viscoelastic Tissue Behavior Based on Experimental Data".

Other mechanical simulations for real time applications based on FEM suggest different methods for optimising performance: Nikitin [106] unifies FEM and pre-computed Green's Functions, Masutani *et al*. [86] describe a surgical simulator based on "FEM and deformable volume-rendering", Miller *et al*. [90] demonstrate an "Explicit Lagrangian FEM". Recent work by Cotin *et al*. [32] focuses on efficient modelling of soft-tissue.

Some success has been achieved in creating real time FEM-based simulations that allow the types of interaction (such as cutting) required in VR surgical simulation applications. Berkley *et al*. [15] have developed a virtual suturing simulation. They use a pre-computed stiffness matrix to improve run-time performance. However, the model cannot be cut or the mesh topology altered since this would require modifying the stiffness matrix which, at the time of writing, was too slow for maintaining real time update rates [15]. Wu *et al*. [163] use a range of optimisation techniques, including some use of the GPU, to accelerate efficient shading to create a deformable soft-tissue model that can be cut.

Clearly FEM is an excellent choice when mechanical behaviour must be accurately modelled. However, adapting FEM so that interactions such as cutting are supported is not trivial.

5.1.2.2 The Boundary Element Method

The Boundary Element Method (BEM) is similar to FEM but uses a set of surface elements rather than volumetric elements to model an object. This simplifies the simulation generally and reduces computational load [70]. However without internal structure BEM is more prone to exhibiting undesirable behaviours like folding. Further, supporting re-modelling interactions like cutting is complicated by the lack of internal structure [95].

5.1.2.3 Green's Function

A Green's function is an "integral kernel that can be used to solve an inhomogeneous differential equation with boundary conditions" [159]. James and Pai [71] describe a method for real time modelling of deformation using pre-computed Green's functions, capacitance matrix algorithms, and wavelets. Performance is accelerated by adapting the resolution of the displacement field that deforms the model. Schoner [129] describes the use of a Discrete Green's Function Matrix to achieve real time simulation of deformable models. Though each of these approaches utilise the Green's function, they are very different. James' method is considerably more complex and achieves higher frame-rates for more complex models, whereas Schoner's work is the converse (simpler and slower). However, neither approach supports interactive mesh-modification such as cutting.

5.1.2.4 Mass-Spring

Mass-spring simulation of deformable objects uses discrete approximations and explicit integration to model material dynamics (whereas FEM uses a continuum model). The mass-spring approach to mechanical simulation uses the relative position of nodes to compute the force of interconnections. The structural representation of the object can be similar (or even identical) to that used in an FEM based simulation. However, rather than solve the dynamics of each element, the force between pairs of nodes can be computed by simulating interconnections as damped springs. According to Hooke's Law, the force from any given spring is proportional to the displacement from its original length ($F = kx$, where F is the restorative force of a spring that is equal to the product of the deviation x of the spring length from its rest length, and the spring constant k). Alone, this formulation results in oscillations between nodes and an inherently unstable system because of the lack of damping. Hence it is typical to use damped springs (Figure 13) where the restorative spring force opposes the rate of change in length of the spring, which thereby reduces or removes oscillations (depending on the amount of damping).

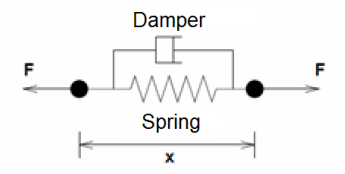


Figure 13: A damped spring cooper [30]

Equation 1 shows Hooke's law with additional terms ($k_d\dot{x}$) expressing the damping force. The damping force is proportional to the rate of change of the spring length $\dot{x}$. Finally, Newton's 1st Law of motion (**Equation** 2) is used to determine the acceleration *a* of a given node due to the total force *F* and its mass *m*.

$$F = k_s x + k_d \dot{x}$$

Equation 1: Damped spring equation

$$a = \frac{F}{m}$$

Equation 2: Newton's first law

A given node within a 3D model will be connected to a number of other nodes. The arrangement of the interconnections and the size of the neighbourhood to which a node is connected will affect the overall behaviour of the mechanical simulation, thereby changing the material properties. Different structural arrangements and their significance have been discussed in more detail in section 5.1.1.

$$\boldsymbol{F} = \sum_{i}^{n} k_s \boldsymbol{x}_i + k_d \dot{\boldsymbol{x}}_i$$

Equation 3: Total force on a node with *i* damped-spring connections

With such a simple basis, a mass-spring system is relatively simple to augment for effects such as cutting. It is also relatively efficient provided the lattice structure used has reasonably low valence. On the other hand the stability and realism of mass-spring based systems is arguably lower than FEM based approaches. However, as I will discuss later there are ways to address these shortcomings (see section 5.2.2).

5.1.2.5 Particle Based and Others

Smoothed particle hydro-dynamics (SPH) is a technique commonly used for modelling fluids. This approach has been adapted by Desbrun *et al*. [36] in order to simulate cutable deformable solids. Unfortunately this work appears only to be demonstrated in 2-dimensions and, looking at more recent publications by the authors, appears not to have been further developed. The concept of particle-based deformable solids may have advantages particularly when handling interactions with fluids (the interaction of interconnected or adjacent systems is simplified when their representations have similarities).

5.2 Cubic Rotational Mass Springs: A New Approach

A number of techniques have been reviewed that are suitable for real time simulation of deformable bodies. Each technique has its advantages: FEM based techniques have excellent stability and analytically verifiable accuracy; mass-spring based systems can be more computationally efficient and are simpler to augment for mesh cutting. Unfortunately, mass-spring based mechanical simulations are also prone to folding. Shear springs (see Figure 9, page 56) with or without additional spring connections beyond the nearest neighbour (e.g. Figure 10, page 56) reduce this problem, however, the additional connections not only result in additional computations, but also increase the number of nodes involved in position update calculations which complicates parallelisation and adds to memory bandwidth requirements. Here I describe a new approach that addresses these issues: Cubic Rotational Mass Springs (CRMS).

Thomaszewski *et al*. [149] showed (in two dimensions) that mass-spring-based systems can benefit from corotational constraints to improve the mechanical stability and realism of mechanical behaviour. I have extended this concept to three dimensions by connecting each node to its 6 nearest neighbours orthogonally (Figure 14). Interconnections are of two types; linear damped springs (Figure 15), and angular damped springs (Figure 16). In three dimensions, this results in 6 linear springs and 12 angular springs (4 per plane) per node. At first impression this may seem excessive since, in the rest position, each of the angles are complementary; if nodes are aligned with the axes then one angle can simply be calculated from the

other. However, under the full range of distortions that occur during deformation each angle must be evaluated individually.

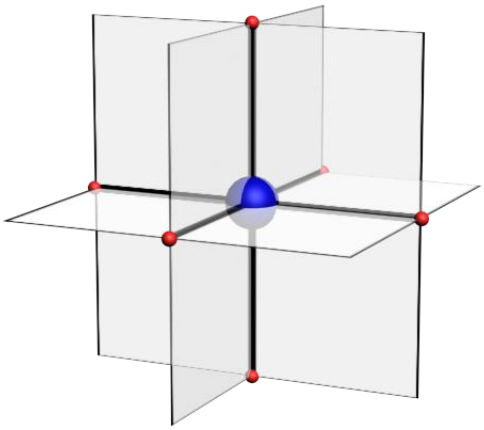


Figure 14: Each node is connected to six neighbours

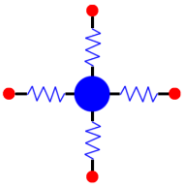


Figure 15: Linear springs

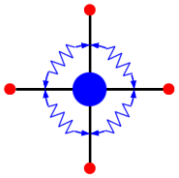


Figure 16: Angular springs

Calculating and applying the linear spring constraints is trivial. However, the calculation of the angular spring constraints is more complex. Figure 17 illustrates the corrective forces applied due to the angular springs for deflection angles of ±45°.

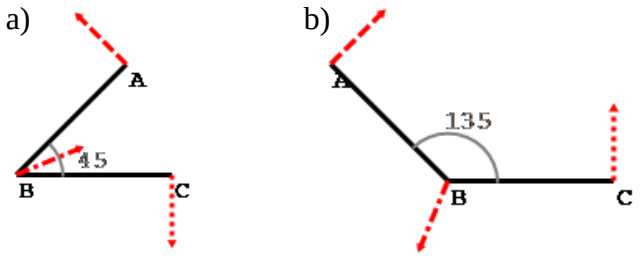


Figure 17: Angular spring corrective forces (90° rest angle)

When computed in 3D-space, forces caused by the angular spring are always planar and acting in directions on the plane formed by the two linear springs (edges). The angular spring deflection δ is computed using the dot product of the edge vectors according to Equation (4). It is important to note that the *acos* function cannot discriminate between angles reflected about 180 degrees and so this equation is only valid in the range $-90° < \delta < 90°$. This causes angular spring deflections outside of this range to incorrectly maintain seek to δ with an offset of 180°. Soft linear springs can remove this problem by enabling edge-springs to

compress or extend without forcing angular joints to deform outside the valid range.

$$\delta = \theta_0 - acos(\overline{\mathrm{AB}} \cdot \overline{\mathrm{CB}}) \quad (4)$$

The direction $\hat{\boldsymbol{d}}$ of the force applied to the node due to an angular spring is computed from the pair of edges (Equation (5)) as shown in Figure 17.

$$\hat{\boldsymbol{d}} = \frac{\overline{\mathrm{BA}} + \overline{\mathrm{BC}}}{|\overline{\mathrm{BA}} + \overline{\mathrm{BC}}|} \quad (5)$$

For simplicity, we combine the deflection δ and the direction $\hat{\boldsymbol{d}}$ to form the term $\boldsymbol{d}$ (Equation (6)).

$$\boldsymbol{d} = \delta\hat{\boldsymbol{d}} \quad (6)$$

Similarly, the linear spring deflection x is calculated from the change in edge length from the rest length of the linear spring L (Equation (7)).

$$x = \frac{|\overline{\mathrm{AB}}| - \mathrm{L}}{\mathrm{L}} \quad (7)$$

The direction of the linear spring force $\hat{\boldsymbol{x}}$ is applied inline with spring itself (Equation (8)).

$$\hat{\boldsymbol{x}} = \frac{\overline{\mathrm{AB}}}{|\overline{\mathrm{AB}}|} \quad (8)$$

We combine the linear spring deflection x and the direction $\hat{\boldsymbol{x}}$ to form the term $\boldsymbol{x}$ (Equation (9)).

$$\boldsymbol{x} = x\hat{\boldsymbol{x}} \quad (9)$$

The total spring force $\boldsymbol{F}$ at a given CRMS node combines the damped spring equation (Equation 1) with the angular and linear spring deflection terms for the 6 linear springs and 12 angular springs (Equation 10).

$$\boldsymbol{F} = \sum_{i=1}^{6}(k_{s1}\boldsymbol{x}_i + k_{d1}\dot{\boldsymbol{x}}_\iota) + \sum_{j=1}^{12}\left(k_{s2}\boldsymbol{d}_j + k_{d2}\dot{\boldsymbol{d}}_j\right)$$

Equation 10: Total spring force at a node

5.2.1 Integration

Simple Euler integration was used for its efficiency and its ability to handle velocity discontinuities caused when handling collisions. Equation 11 shows how the current position $\boldsymbol{p_{curr}}$ and current velocity $\boldsymbol{v_{curr}}$ are updated according to the time step Δt, acceleration $\boldsymbol{a}$, and previous position $\boldsymbol{p_{prev}}$ and velocity $\boldsymbol{v_{prev}}$.

$$\boldsymbol{p_{curr}} = \boldsymbol{p_{prev}} + \boldsymbol{v_{prev}}\Delta t$$
$$\boldsymbol{v_{curr}} = \boldsymbol{v_{prev}} + \boldsymbol{a}\Delta t$$

Equation 11: Euler integration

5.2.2 Stabilising the System

Mass-spring systems are inherently unstable under high loading. When stress or strain forces are large, one common problem is referred to as overshoot. When explicit integration schemes are used and if springs are under high loads, overshoot can quickly produce an unstable system which, without special measures, will cause nodes to oscillate and quickly accumulate kinetic energy that can result in "jiggling" or potentially "explosion" whereby the system becomes unstable.

5.2.2.1 Limiting System Kinetic Energy

One approach to extending the maximum deformation for which the system can remain stable is to reflect overshoot according to the projected position of a node given the integration period and the current spring stiffness. Alternatively, system stability can be maintained for higher forces by adding a speed-dependent kinetic damping. This will act to reduce the kinetic energy in the system (the motion of the nodes). CRMS uses this technique increase the stability of the system. When set too high, the system-kinetic-energy constant ('k_{sys}') gives similar behaviour to an overdamped system. Since use of this approach relies only on that rate of change of the linear spring it is very efficiently applied in parallel.

Finally, in order to further enhance the stability of the system a simple velocity limit was used when forces and consequent acceleration of nodes relative to one another is high, the relative velocity of nodes quickly increases. Increased damping is will lessen the problem, but when damping is set too high the system becomes slower and less responsive (like a sponge filled with oil). Use of a

velocity limit increases the stability of the system for a wider range of mechanical characteristics without over-damping the system.

Both methods of extending the stability of the system are applied efficiently within the GPU implementation resulting in only a few additional instructions per node update.

5.2.3 Performance Optimisations

CPUs use branch prediction and elaborate caching strategies to hide memory latencies to optimise execution speeds of a few threads. Conversely, GPUs are throughput-oriented with smaller caches and are optimised to execute thousands of threads in parallel. Further, GPUs have comparatively more processing capability and less memory (and bandwidth) per processing unit than CPUs. Therefore, in order to optimise software for efficient execution on the GPU the developer must take steps to minimise the impact of memory latencies and limited bandwidth per processing unit. Two key strategies were employed in developing a mechanical simulation that maximises performance on the GPU:

1. Compute rather than look up data wherever possible.
2. Exploit coalesced global memory usage.

5.2.3.1 Implicit Node Addressing

Updates to the mass-spring system use the relative position of each node's neighbours. Mass-spring systems based on irregular tessellations must explicitly address nodes by storing pre-computed adjacency information. Explicit addressing can also improve performance by removing the need to repeat nearest neighbour searches at run time.

By using a regular cubic lattice structure, CRMS avoids any expensive local searches completely because adjacent nodes are implicitly addressed; Addresses are calculated rather than read from global memory, which reduces global memory access, instead adding a small number of computations for calculating the address of adjacent nodes. Consequently, memory-related bottlenecks are avoided, resulting in better utilisation of the GPU's compute capabilities.

In summary, CRMS resists folding while minimising the number of nodes involved. The memory bandwidth required during updates is reduced by minimising the number of connected nodes, thus allowing better utilisation of GPU resources than existing methods. Moreover, the simplicity of the mass-spring approach provides a strong basis for the tissue simulation and allows for cutting and volumetric tissue removal (refer to Chapter 7). The remainder of this chapter outlines the performance of the system on the GPU and demonstrates its extensibility by applying a range of specialisations to enhance its application in VR medical training simulations.

5.3 Demonstration

The CRMS runs efficiently on the GPU by splitting update computations into one GPU thread per node. The system can be adjusted to exhibit a wide range of mechanical characteristics. Material stiffness can be varied from soft tissue through to stiff, almost-hard tissue. Damping can also be tuned to vary the "responsiveness" of the simulation. Critically damped or slightly under-damped springs will result in jelly-like behaviour. Conversely, overdamping produces less responsive tissue, which reverts to its original shape more slowly. Additionally, the separation of parameters controlling the properties of angular springs versus linear springs enables the simulation to produce new behaviours. For example, stiff angular springs and soft linear springs will result in tissue that is compressible but which resists bending and shear deformations. Soft angular-springs and stiff linear-springs cause the lattice to collapse (fold flat around itself). However, when subject to strain (extension) this configuration exhibits behaviour which can be likened to woven fabric; resistant to strain and low-resistance to shear loads.

CRMS is capable of simulating grid resolutions up to 64 x 32 x 32 at interactive rates (update rates above 30Hz) (Figure 18). Relatively long and thin structures can be simulated when stiff springs are used (Figure 19).

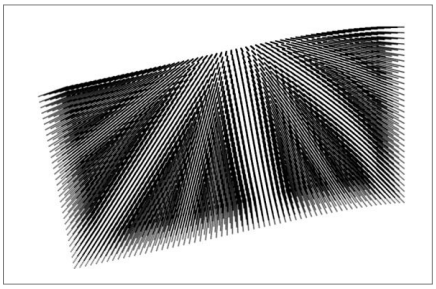


Figure 18: High-resolution square beam (64x32x32 nodes)

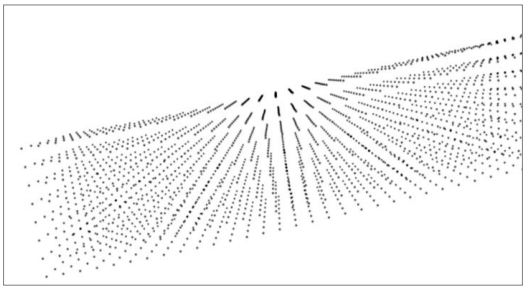


Figure 19: Long stiff beam (32x8x8 nodes)

CRMS will twist in response to torsional forces (Figure 20). Individual cells (cubes of eight nodes) can be deflected past 22.5 degrees without folding resulting in stable simulation of large deflections (Figure 21).

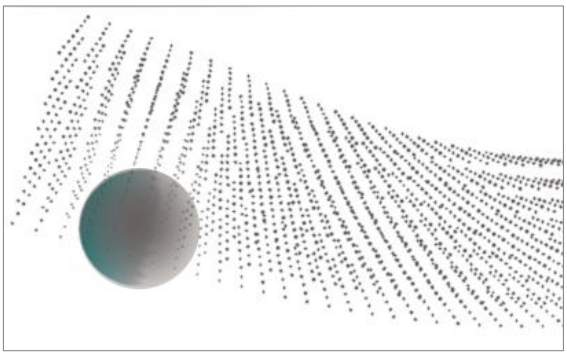


Figure 20: Twisting deformation of beam

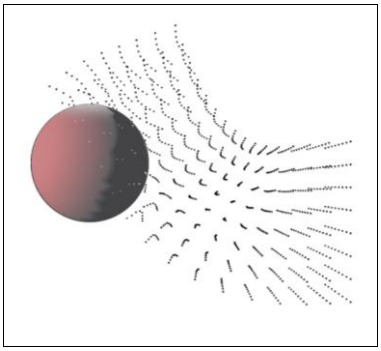


Figure 21: Large rotational deformation

Cloth can also be simulated efficiently with a single-layer of CRMS. The lattice cannot be punctured or popped-through (see 8.6.1.1) even when stretched. Under tension, the rotational springs do not significantly alter the behaviour of CRMS (Figure 22 and Figure 23).

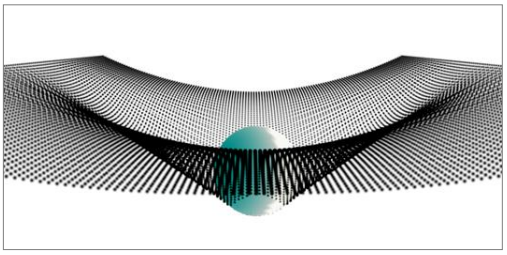


Figure 22: Cloth with linear springs only

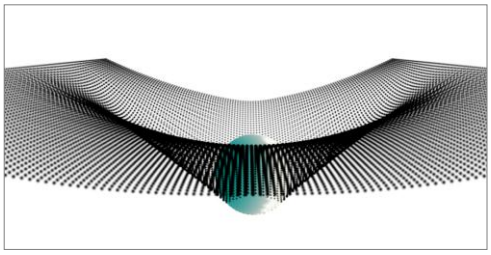


Figure 23: Cloth angular and linear springs

The effect of the angular springs is far more significant when the CRMS cloth is draping or hanging (Figure 24 and Figure 25).

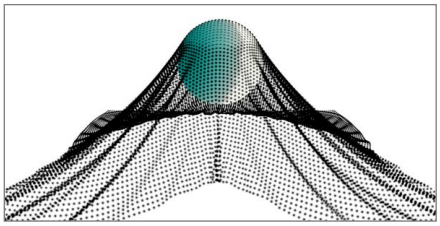


Figure 24: Cloth with angular and linear springs

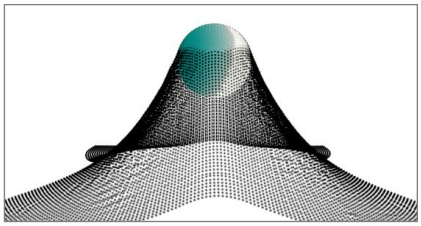


Figure 25: Cloth with linear springs only

Unlike many cloth simulations, CRMS allows multi-layered cloth to be simulated. This creates a mattress-like effect that is useful for simulating layers of tissue (Figure 26).

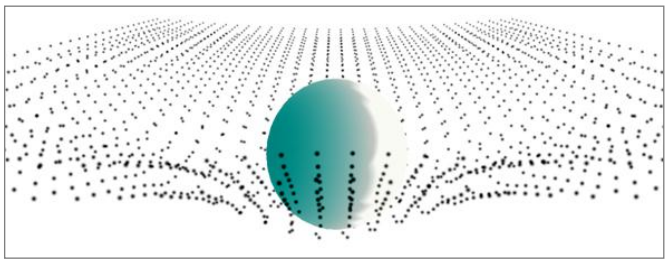


Figure 26: Two layered cloth with angular and linear springs (32x32x2)

5.3.1 Improving Performance with Memory Access Coalescing

Although CRMS minimises the usage of global memory, data such as node positions must be stored there because of its size and the access patterns required. Accessing global memory is relatively slow and consequently will limit performance by stalling processing. Global memory access coalescing can be used to perform a staggered pre-fetch of data stored in global memory, thereby hiding latency and increasing overall performance. Consequently, global memory coalescing is one of the most significant ways to increase performance on the GPU [109].

In order to use global memory access coalescing, certain criteria must be met. Recent changes to CUDA expand the range of access patterns that will be coalesced [108]. However, the simplest access pattern is the one targeted by CRMS. This pattern is achieved when each thread accesses global memory with a consistent address offset to temporally adjacent threads. In order to fulfil CUDA's requirements for coalescing, global data is stored as arrays with the pre-requisite types as specified in the Programming Guide [108]. Each node update performs six

lookups with typically consistent address offsets (the exception being near mesh boundaries), which allows updates to occur substantially faster since groups of sixteen threads (termed “warps” in the CUDA context) can access 64 byte blocks of memory using one instruction for the entire group.

5.4 Extensibility

It is particularly important that the tissue simulation be capable of mimicking the behaviours of real tissue. Many tissues can exhibit plasticity and visco-elasticity. It is normally difficult to create a mechanical simulation that can mimic these properties and requires considerable effort and expertise to model them accurately. However, it is more important that the simulation be compelling rather than absolutely accurate, provided that immersion is not inhibited and properties are not misleading it is sufficient to use techniques which deliver a compelling illusion. Here I demonstrate the versatility of the system design by describing a range of effects that significantly alter the behaviour of CRMS with minimal overhead.

5.4.1 Approximating Plasticity

Plasticity and visco-elasticity can be approximated in CRMS using a simple heuristic that modifies the spring rest-length as a function of current spring length (proportional to stress/strain). When a compression threshold is exceeded (minimum spring length), the spring’s rest length is scaled with time. Without a recovery function, these changes to the rest length modify the rest-shape of the mechanical simulation, thereby mimicking plastic behaviour. By adding the recovery mechanism, visco-elastic behaviour is approximated.

5.4.2 Anisotropy (Axial bias)

Many tissues exhibit different properties when probed in different directions. The mechanical characteristics along one axis can differ significantly from the mechanical properties along another axis. Muscle tissue is an obvious example, but even apparently homotropic tissue can also exhibit anisotropic mechanical behaviour due to vasculature or other structures that are nominally aligned. Consider the kidney. Nephrons are radial structures that lead to different mechanical characteristics in radial and tangential deformations.

Anisotropic characteristics can be added to the mechanical model by adding axial stiffness and damping modifiers. A normalised 3-dimensional vector can be used to modulate the stiffness and damping of a volume at the same resolution as the mechanical simulation. For example, to simulate the anisotropic behaviour of a tensed muscle, a larger multiplier in the axis of the muscle-fibres would increase the spring stiffness while perpendicular to this axis it will decrease.

5.4.3 Approximating Nonlinearities using Fluid Dispersion

Tissue tonometers collect information about the mechanical characteristics of tissue. Tissue that is recovering from damage, such as burns, exhibits a range of mechanical characteristics "from the initial fluid-rich stage through the fatty middle stage to the fibrous end point stage", which can be differentiated using tonometry [11]. This illustrates the diverse mechanical properties, even within a single tissue type, that living tissue can exhibit and also has provided the inspiration for a new technique that can be readily incorporated into CRMS.

Evaluating the change in volume of CRMS cubes can be efficiently computed in parallel. This change in volume can be used to compute the pressure within each set of nodes forming a (deformed) cube. The pressure then modulates the spring stiffness. Diffusion can also be efficiently simulated using a diffusion rate to propagate the pressure into adjacent cubes. Additional hardware acceleration can be achieved by leveraging the raster-operations capabilities of the GPU to propagate pressure values throughout the lattice by storing values in a 3D texture and down-sampling them. The effect of the down-sampling thereby approximates fluid dispersion.

5.4.4 Heterogeneity (Variability)

Each of the augmentations to the mechanical simulation already described in this section can be applied globally or locally per-node. When applied locally it is possible to vary the mechanical properties throughout the material. In so doing, the system can model a diverse variety of tissues within the same instance of the tissue simulation. This enables the system to describe anatomies at various scales with detail only limited by the number of nodes it is possible to simulate on the given

hardware. Hence, CRMS can scale with advances in hardware to create more detailed and realistic simulations without having to re-engineer the system.

Furthermore, it is simple to define mechanical simulation parameters on a per-node basis to achieve diverse mechanical properties within a single instance of the CRMS. The main limiting factor when adding per-node variables is that reading these values from memory will eventually saturate the available bandwidth. This limit is however reached much later than if the approach were used in existing systems because of the reduced valence of CRMS.

One way to further reduce bandwidth usage is to use a physical property palette. A small number of parameter sets can be used to describe a limited number of primary tissue types. Then, rather than needing to read $n_1 \times n_2$ values from memory (where n_1 is the number of parameters and n_2 is the number of nodes) the GPU need only read the palette blend weights for each node and store the palettes of parameters in local memory.

5.4.5 Animation

Traditionally, animation uses poses and keyframes. Models are put into poses using joints and blend weights applied to the rigid model. Animating the behaviour of the mechanical simulation can use the same approach to create different rest-shapes that the system will attempt to revert to.

A flexing muscle not only changes shape, its mechanical properties also change. In order to simulate this behaviour, the mechanical simulation parameters can be animated. There are different options available to provide this functionality. For example, linear spring stiffness and damping can be animated independently at specific nodes of the simulation.

The mechanical parameters can also be animated. Matrix palette skinning uses a palette of matrices for each vertex [50]. Rather than use the palette of matrices to drive the position of vertices directly, instead CRMS could use a palette of mechanical properties and a time series of blend weights to describe animated changes to mechanical properties of the simulation.

This idea is particularly exciting when considering its ability to enrich the medical simulation. For example, triggers such as cuts to a specific part of the simulation can be used to trigger transitions to new mechanical property presets. In this way a pulsing artery can go flaccid (and stop pulsing) if severed or if blood pressure in the patient model is reduced.

5.4.6 Tearing

Since each node of the simulation is updated independently, the forces within the system caused by a given node of the simulation can easily be masked or omitted from the simulation. By introducing a tearing threshold, CRMS can control the force needed to cause a node's forces to be masked, which allows the mechanical simulation to be torn. Implementing tearing in this way does not preserve volume (since nodes along the tear are removed rather than split), but it is simple and robust and produces a compelling effect particularly when using a high-resolution mesh.

5.5 Summary

CRMS simulates diverse material characteristics and capabilities normally only present in offline (non real time) approaches or non-cutable FEM-based methods by efficiently leveraging the massive parallel compute capability of the GPU to maintain the high update rates necessary for haptic interaction (Chapter 8). The impact of limited memory bandwidth is reduced by using a cubic lattice that enables fewer adjacent node lookups, which are further accelerated using coalesced memory access. It also includes a range of enhancements that extend the range of potential uses of the system. Finally, CRMS has been engineered to facilitate efficient integration into the complete tissue simulation system, which is detailed in subsequent chapters.

Chapter 6. Interactive Marching Tetrahedra

The CRMS component (described in Chapter 5) uses a simple cubic grid that, without further enhancement, cannot produce a high quality visualisation. Further, the CRMS grid is intended to be scalable so that grid cells can be used to represent whole organs efficiently using minimal processing resources. This allows multiple instances of the CRMS system representing different structures to coexist within a single simulation. At these resolutions, details of finer structures are lost. This chapter describes a new system to add high-resolution detail and produce a standard 3D mesh output that can be readily visualised with common rendering algorithms (Aim 5). Moreover, this high-resolution 3D overlay is optimised to support cutting and ablation with minimal processing (Aim 3).

Having described a system that delivers a plausible real time soft-tissue mechanical simulation (Aim 2), a method of visualising it is required. The simplest solution is to directly visualise the mechanical simulation. However, this requires a high-resolution in order to achieve a reasonably realistic render. Moreover, it presents problems in conditioning the grid surface so as to cater to the rendering algorithm's requirements. For example, in order to compute the lighting of a surface, high-quality rendering algorithms require surface normals. Since normals to the surfaces of the CRMS' cubic grid are aligned with each of three orthogonal axes, any rendering algorithm would need to overcome this or else the grid would not appear smooth. A more general approach is required that can readily be rendered with realistic lighting effects (Aim 5).

Several existing tools are capable of coupling a high-resolution render mesh with a mechanical simulation, for example Nvidia's PhysX. However, as Chapter 3 points out, none of the existing tools support the types of interactions

required in VR surgical simulation. In particular, in addition to adding high-resolution detail and realistic rendering, additional functionality to enable cutting and ablation is also required. Existing approaches commonly insert additional vertices into the mesh to create the new edges created by the cut. This can be difficult to maintain, particularly when cuts are made into already cut edges. If not done carefully, inaccuracies accumulate, resulting in a reduced quality mesh. These difficulties can be overcome by using a uniform approach to surface generation that is capable of representing cut and ablated areas in the same manner as unmodified surface areas. However, this may remove some opportunities for surface optimisation.

The technique described herein is based upon technology originally developed to process medical imaging data. Medical scans such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) output volumetric datasets. These datasets are large, often consisting of tens of gigabytes. In order to facilitate visualisation and reduce the amount of data needed for storing and sharing medical scan data, techniques to generate surface meshes from the scan data have been developed. Marching cubes is one of the most widely used methods.

6.1 A Review of Marching Algorithms

Lorensen and Cline first presented the marching cubes algorithm to create triangle meshes from medical imaging datasets [82]. The marching cubes algorithm is initialised by establishing a grid of points over the region of interest. Each point is assigned a state according to the region of interest. The algorithm then iterates through the grid as cubic elements (voxels), a set of eight points, and provides a solution for triangulation for each element based on the states of the cube. By symmetry, the triangulation solution provided is reduced to 14 patterns that are stored as a lookup table. However, the problem inherent to the marching cubes algorithm is the generation of inconsistent topology, or holes, under certain states [38].

The inconsistency generated by the marching cubes algorithm can be resolved by subdividing the cubic elements into tetrahedral elements. However, as

reported by Zhou *et al.* [165], there are two methods for subdividing the voxels into tetrahedras, which results in differing surface topology that depends on the initial subdivision methodology. Zhou *et al.*, propose a cubic interpolation to resolve the inconsistency, but with a lookup table of 59 patterns, the solution is far more complex than the original marching cubes algorithm. To alleviate the complexity, Chan *et al.* [25] developed a tetrahedral tessellation scheme that retains the advantages of Zhou *et al*'s but with fewer patterns.

6.2 Marching Tetrahedra

Chan *et al's* tetrahedral tessellation scheme is based on the body-centred cubic lattice. As shown in Figure 27, the tessellation scheme is applied to a lattice with an interpolated grid point in the centre of each voxel. Twelve tetrahedra are evaluated for each voxel to form the isosurface (3 orthogonal sets of 4). This results in consistent topology with but has the disadvantage of increasing the number of triangles used when compared to the original marching cubes algorithm. However, since the scheme does not lead to any ambiguities in generating the surface mesh, it provides an ideal basis for the overall framework to re-mesh the surface interactively.

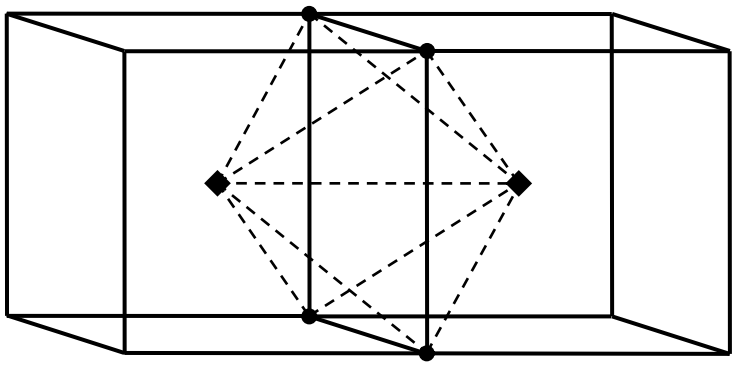


Figure 27: Chan et al's marching tetrahedra tesselation scheme

An isosurface is generated from a grid of voxels by iterating through all voxels and generating surface triangles wherever these voxels span an edge. For each voxel, three sets of four tetrahedra are evaluated. Cube corner voxel values are interpolated to give the body-centre voxel. Individual voxels are evaluated to determine which of eight cases (7 edge cases or empty) has occured (Figure 28). Zero, one or two triangles per tetrahedra are then added to the isosurface (triangle

mesh) model. Vertex positions are moved along tetrahedra edges according to voxel values to improve the accuracy of the generated isosurface.

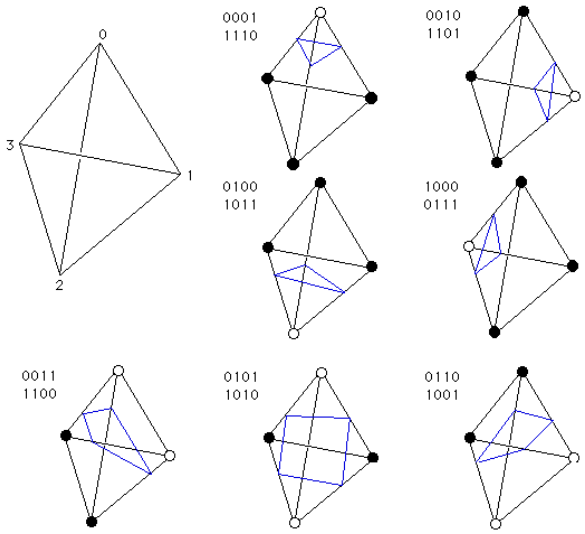


Figure 28: The seven edge cases for the marching tetrahedra algorithm [17]

6.3 Interactive Marching Tetrahedra: A New Approach

This component (IMT) increases the resolution of the CRMS model and outputs surface mesh geometry in a format that can be rendered using common lighting algorithms. Equally importantly, it provides an efficient method for adding support for interactive cutting and volumetric tissue removal (ablation).

IMT employs the same tessellation scheme as the marching tetrahedra algorithm developed by Chan *et al.* and introduces an efficient method for updating changing sub-volumes. This enables IMT to sustain the high update rates required for high-quality haptic rendering for an interactive model that can accumulate any number of cuts.

A common approach to improving the performance of marching algorithms is to use two passes. The first pass identifies boundary voxels, and the second computes the surface. The identification of the boundary voxels can be accelerated using spatial partitioning schemes similar to those employed in real time collision detection, such as binary spatial partitioning. By reducing the computational cost of handling empty voxels, overall performance is substantially increased. However, although faster than directly applying a marching algorithm, the resolution of the volume that can be surfaced at interactive rates is limited.

The processing required to perform interactive updates is proportional to the volume of interaction; the volume of the cutting part of a blade, or the volume of tissue removed during a single update. Since these volumes are typically far smaller than the tissue being modelled, the user only interacts with a very small percentage of the entire model at a time. This approach substantially reduces the processing required, allowing a much higher resolution volumetric model to be used. The system's performance is determined by the computational cost of updating small volumes, rather than the entire model.

Since the volumetric model can now be dynamically modified, the surface model's complexity varies. Normally the changing surface mesh would require vertices of the mesh to be appended and deleted depending on whether the surface area of the model has increased or decreased. In turn, the required vertex buffer memory changes. Since GPUs require vertices to be stored in contiguous memory, an efficient mechanism for handling the changing vertex count is required.

Computer graphics APIs (section 3.1.4) support a number of methods for presenting geometry to the graphics pipeline. Use of an index buffer reduces the memory needed by allowing multiple triangles to re-use any given vertex, and it also provides the abstraction layer needed for dynamically re-sizing the mesh since the vertex and index buffers can be pre-allocated.

The computational costs associated with dynamically managing new memory allocations during run time are avoided by using pre-allocated graphics memory to store vertices and indices of the model. As triangles around the volume of interaction are moved they are deleted and recreated in their updated location. When triangles are associated with a changing part of the volume they are first marked for re-use and then overwritten with the new triangles. All indices of triangles marked for re-use are changed to reference a single point to prevent them from being rendered. Though not as neat as proper deletion, this avoids moving index and vertex data and in so doing allows more detailed models to be cut and ablated at interactive rates.

6.3.1 Improving the Mesh Quality

The output of the marching algorithm is an unordered set of triangles, which is far from optimal for rendering since it contains approximately three times the number of vertices and vertex normals as would be required by the same mesh stored as a triangle strip. Since the volume is a closed surface, each triangle vertex is shared by at least three other triangles. The un-optimised output of the marching algorithm can be displayed directly but doing so is a waste of processing. A mesh optimisation stage is introduced to improve the mesh quality.

6.3.1.1 Mesh Optimisation

The mesh optimisation stage introduces a new processing stage to efficiently improve the quality of the marching algorithm's output. Output is optimised by reducing the amount of redundant data used to represent the model. The optimisation also improves the quality of the mesh by removing tiny triangles formed by the marching tetrahedra algorithm.

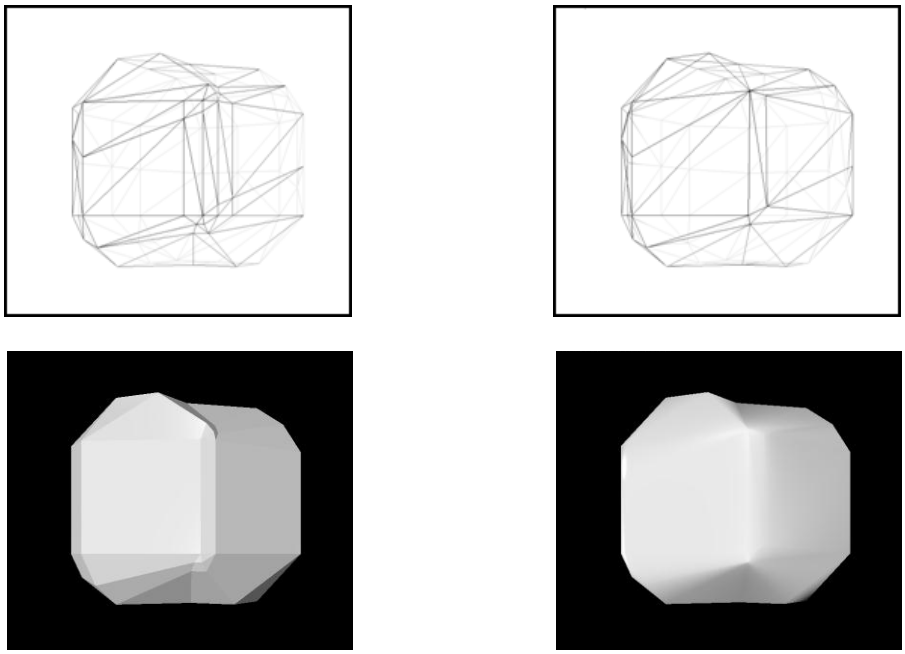


Figure 29: An IMT generated 4x4 voxel "ball" (left: unoptimised, right: optimised)

Coincident or tightly grouped vertices are replaced with references to a single vertex, which reduces the total number of vertices used. Referencing vertices in this way also makes identifying tiny triangles trivial since they now reference the same vertex more than once. Hence, these triangles can be efficiently removed

from the render model. Figure 29 shows the effect of the optimisation stage on a simple model. The un-optimised model is composed of 144 triangles that are explicitly stored as 432 vertices (3 x 144) whereas the optimised model is comprised of 108 triangles and only 70 vertices. Notice also that the optimised model appears smoother since vertex normals are combined as a weighted average (according to triangle area) of the triangle normals.

Processing is required to perform updates to the model by using spatial hashing. This greatly simplified proximity tests that are performed when new vertices are added. Reference counts are maintained in order to identify when a given vertex can be removed from the rendered set, and its memory location marked for re-use.

6.4 Demonstration

The IMT approach has been used to create models of up to 256 cubed voxels (approximately 16.8 million voxels) that can be interactively cut and ablated (Figure 30). The maximum supported resolution is a limitation of the data-structures used rather than processing load. The system is currently implemented to create basic parametric shapes, however, it is relatively simple to base the model on any volumetric dataset including medical scan data (Figure 31).

Figure 30: Raw output of the IMT algorithm (256^3 voxels)

Figure 31: The IMT algorithm initialised using CT scan data of a tooth

The optimisation stage efficiently removes small and excessively thin sliver triangles from the model (Figure 32). Normals can optionally be smoothed to improve the final render quality (Figure 33). The model shown in Figure 32 was originally composed of 8,794 triangles (blue wireframe mesh). After the optimisation stage the model consisted of 5,418 triangles (a reduction of 38%).

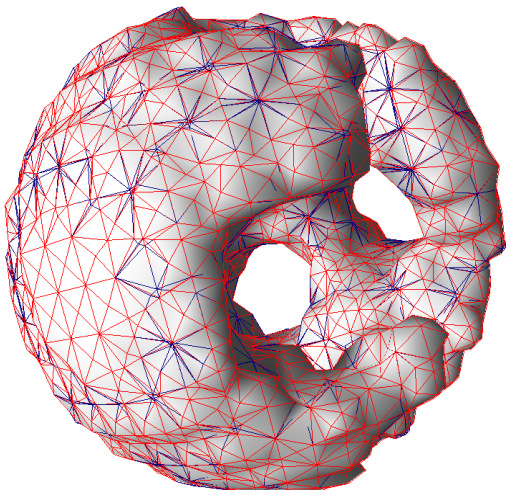


Figure 32: Optimised mesh (red) versus un-optimised (dark-blue) (32 cubed voxels)

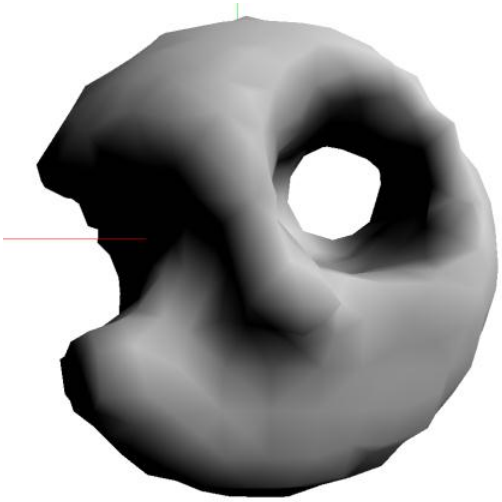


Figure 33: The final result: optimised and smoothed (32 cubed voxels)

6.5 Summary

The IMT component enables efficiently interactive isosurfacing optimised for high-quality rendering. The resolutions supported are substantially higher than what would otherwise be possible with existing methods. This improvement is achieved by minimising the complexity of updates to the model geometry.

Chapter 7. Integration of Components

CRMS and IMT alone are not capable of delivering a tissue simulation that users can cut and ablate as required (Aim 3). Like some existing soft-body simulations, while the CRMS mechanical simulation effectively approximates the behaviour of real tissues in response to user interaction, it cannot be cut. At the same time the IMT component efficiently provides a new method for interactive cutting and volumetric remodelling but is not deformable. This chapter describes how these two components have been combined to create a cut-able and ablatable soft tissue simulation framework (TSF) that realises a critical requirement of VR medical simulations that is not currently accessible to medical simulation software developers.

Having developed efficient solutions for the two previous components; 1. Cubic rotational mass-springs (CRMS), and 2. Interactive marching tetrahedra (IMT) it was important to retain efficiency by avoiding the introduction of processing overheads (and memory bandwidth overheads) that undermine the efficiency of the system as a whole. Further, the integration must result in a robust whole that is both versatile and stable. Integration of these components required development of specialised algorithms to ensure that the visible model and the mechanical behaviour tightly correlated, particularly during cutting and ablation.

The IMT component outputs a high-resolution triangle mesh that, without integration, remains separate from the deforming CRMS grid and is therefore rigid and non-deformable. This provides a higher quality render by allowing the use of a 3D model that is optimised for display (rather than using a single mesh that fulfils requirements of both rendering and mechanical simulation). In order to deform the IMT mesh it must be coupled with the CRMS grid such that deformation of the

CRMS grid suitably deforms the IMT generated mesh. Furthermore, the CRMS grid must be maintained such that any tissue removed via the IMT component is reflected in the mechanical simulation. Finally, the deformation of the IMT-generated mesh introduces an alignment problem; user interactions with the IMT volumetric model must now correlate with the deformed coordinate space. This chapter describes how each of these issues was addressed.

7.1 Deforming the IMT Mesh

The separation of high-resolution render geometry (IMT mesh) from a coarser mechanical simulation (CRMS lattice) is not new [42]. Sometimes referred to as cartoon meshing, mesh coupling allows a smaller number of reference points to warp the coordinate space of a higher resolution model (Figure 34), which reduces the computational cost of rendering a detailed deformable model by allowing the more computationally intensive mechanical simulation to use a lower resolution.

Coupling is achieved by aligning the IMT mesh coordinate space with the mechanical simulation, then scaling it so that the extent of the two coordinate spaces match. For each vertex of the IMT mesh, an index that identifies the cube of the CRMS lattice that contains the vertex is stored. This reference cube then defines a local coordinate space within which the vertices are located. As the mechanical simulation deforms (Figure 35, white dots), the coordinate frame deforms, thus deforming the set of rigid IMT vertices located therein (Figure 35, shaded).

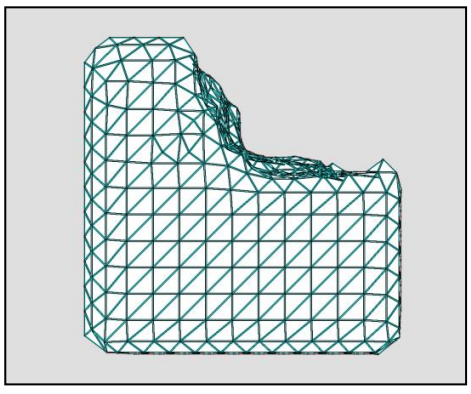


Figure 34: Wireframe IMT mesh

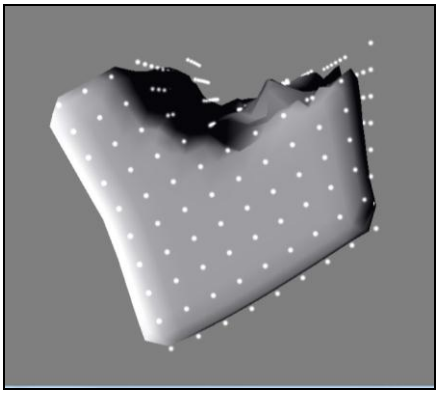


Figure 35: Deformed IMT mesh

When the IMT-generated mesh changes, vertices of the render mesh are moved, deleted and inserted. Thus the mesh coupling system requires special handling so that modified vertices are always coupled with the correct cube in CRMS. This is relatively simple and consequently does not significantly impact performance because only minimal processing is required (one instruction to map the vertex to the CRMS coordinate space, plus one modulo operation to determine the cube index).

The processing required to update coupled vertices depends on the number of new vertices created, which is dominated by the cost of updating the optional mesh smoothing and optimisation stages of the IMT component. This cost limits the maximum number of new vertices that can be created per frame. A few thousand changed vertices per frame can be updated without impacting refresh rates.

7.2 Cuts and the Coupled System

Cutting the visual model (IMT mesh) is supported by removing IMT voxels in the shape of the cutting instrument's blade. However, without further enhancement the system will not behave correctly since cuts in the IMT mesh are not represented in the CRMS lattice.

IMT vertices are coupled to CRMS nodes that span the IMT mesh surface. CRMS nodes that are internal are differentiated from external (empty) CRMS nodes using the IMT voxel data. As cuts are made, CRMS nodes are deleted if less than 12.5% (⅛) of the IMT voxels are occupied (Figure 36 and Figure 37; deleted CRMS nodes indicated by ○, and undeleted nodes by ●). Applying this criteria ensures that the coarser CRMS lattice matches the shape of the IMT mesh as closely as possible. Marking CRMS nodes as deleted requires insignificant processing or bandwidth. Furthermore, no additional processing is required to skip updates to deleted CRMS nodes since updates are performed in parallel on the GPU. Hence, the only additional processing incurred by cuts is the increase in rendering processing due to the increased complexity of the IMT mesh.

Higher deletion thresholds result in CRMS nodes being deleted when more IMT voxels remain occupied. Figure 38 and Figure 39 show the effect of a higher

threshold (¼) on the two cases illustrated in Figure 36 and Figure 37 (notice that additional CRMS nodes are deleted). In particular, Figure 38 highlights (white arrow) a particular CRMS node that is deleted despite being proximal to a significant number of occupied IMT voxels. This CRMS node occupies one of the lower right corners of the cube that is used to locate the local IMT mesh vertices.

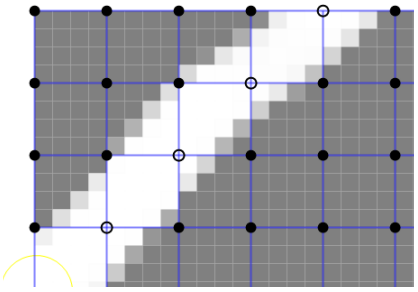


Figure 36: Diagonal cut where CRMS nodes less than an eighth occupied are deleted (IMT:CRMS ratio of 4^3:1)

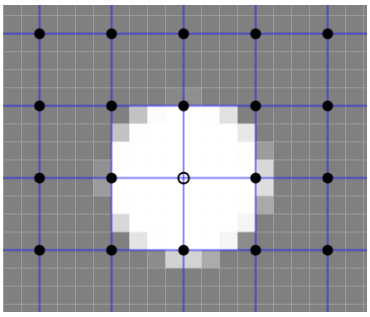


Figure 37: Hole where CRMS nodes less than an eighth occupied are deleted (IMT:CRMS ratio of 4^3:1)

The coupling system uses undeleted CRMS nodes to infer the position of missing (deleted) CRMS nodes that are needed to deform the IMT mesh. The coupling requires at least 6 cube corners to position IMT vertices (refer to Appendix A). Further work is required to increase the coupling system's tolerance of missing CRMS nodes so that higher deletion thresholds can be used. One possibility is to use adjacent non-deleted CRMS nodes to infer missing node locations.

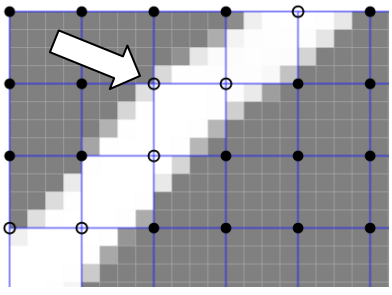


Figure 38: Diagonal cut where CRMS nodes less than a quarter occupied are deleted (IMT:CRMS ratio of 4^3:1)

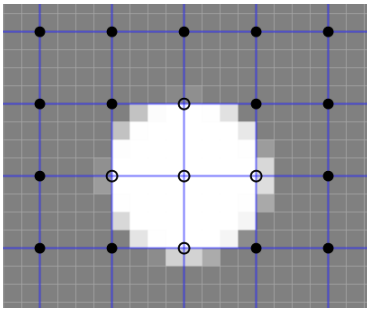


Figure 39: Hole where CRMS nodes less than a quarter occupied are deleted (IMT:CRMS ratio of 4^3:1)

If the resolution of the IMT voxel dataset is too high compared to the CRMS lattice resolution, the visual and mechanical systems no longer reliably correlate; Cuts in the IMT model may not be reflected in the CRMS lattice. Figure 40 shows a

cut in the IMT model that is not represented in the CRMS model. The minimum viable cut width can be reduced by increasing the CRMS lattice resolution (Figure 41). Alternatively, a higher CRMS node deletion threshold can be used (Figure 42).

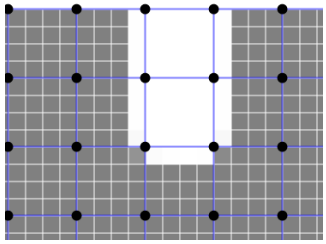


Figure 40: A small cut to the IMT mesh that has not cut the CRMS model

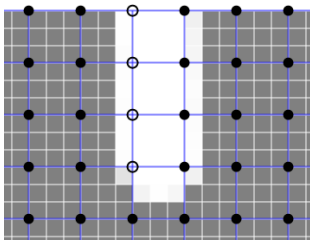


Figure 41: Increasing the CRMS model resolution allows finer cuts

One of the objectives of the TSF is to enable the use of a relatively coarse CRMS lattice to reduce the processing needed to perform updates to the system. Ideally very fine cuts (Figure 43) would result in cuts to the CRMS model. This can be accomplished by breaking individual spring connections between CRMS nodes (rather than deleting them entirely). However, this is outside the scope of this thesis and left for future work.

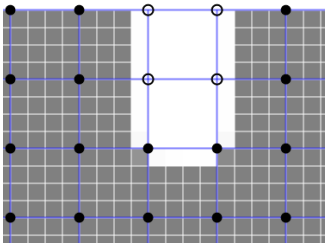


Figure 42: Increasing the CRMS node deletion threshold allows finer cuts

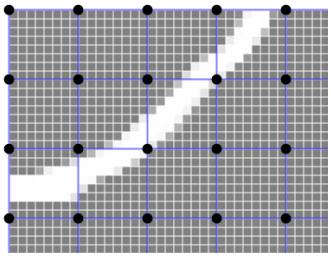


Figure 43: A cut through the IMT model only (IMT:CRMS ratio of 8^3:1)

7.3 Collisions

In order to interact with the model, TSF must detect when user-controlled instruments touch (collide with) the model. Rigid models can relatively easily be optimised for collision detection using spatial partitioning schemes or other methods (see section 3.1.6). Usually a broad-phase collision detection is used to efficiently reduce the search-space, then a narrow-phase collision detection identifies individual intersecting primitives (triangles, edges, or vertices).

While rigid structures can be pre-sorted into various data structures that facilitate fast and efficient lookup of small sets of collision candidates, deforming models are more difficult to optimise for collision queries. When a model deforms there is no guarantee that the initial location of a given vertex or triangle is close to its original location. Hence the data structure must be updated regularly, or an alternate method of optimising collision queries must be used.

User interactions with tissue during surgery typically use precision instruments that impact small parts of the overall volume. TSF exploits this by mapping the model of the user controlled instrument into the deformed coordinate space of the CRMS model (rather than the converse). This strategy reduces the complexity of collision detection by making processing dependent upon the small number of vertices in the interactive part of the user-controlled instrument rather than the entire tissue model. Once the intersecting nodes of the CRMS component are identified, the narrow phase collision test is then conducted using the coupled (deformed) IMT mesh vertices. Since each vertex of the IMT mesh is mapped (coupled) to a CRMS node, the reactive forces (due interactions of the IMT model with the stylus) are also mapped efficiently into the mechanical simulation.

In summary, collision detection is performed as follows:

1. (optional) Use a long stride (4 or more nodes) to identify the nearest CRMS nodes
2. Test all candidate CRMS nodes for intersection with the instrument OBB (oriented bounding box)
3. Look up coupled IMT mesh vertices for colliding CRMS nodes
4. Perform per-primitive (narrow phase) collision detection

This approach efficiently avoids much of the processing that would otherwise be necessary to identify colliding primitives.

7.4 Demonstration

The tissue simulation system described in this thesis successfully delivers a new type of interactive soft-body simulation capable of simulating a diverse range of material properties. The combined IMT and CRMS systems enable the user to volumetrically remove tissue from anywhere within the tissue. Moreover, the mechanical simulation component maintains consistency with the visual

representation to allow parts of the tissue in any shape to be cut away from the original volume. Any shape can be used as the initial tissue-model. Figure 44 below shows the tissue simulation as it is cut and deformed. Figure 44: a) A cube of tissue (attached along the right edge to an invisible wall) is deforming slightly under the effect of gravity. b) A small stylus has removed tissue from the top of the model. c) A haptic stylus controls a yellow ball that causes the remaining tissue to deflect as it is pressed against the left edge of the tissue model. d) The stylus controlled ball is pressed down on the tissue model from above causing the cut to open up and the entire model to deform. e) More tissue is removed leaving a complex yet smooth shape with some thin pieces still remaining. f) The thin pieces of tissue are more easily deflected using the stylus controlled ball.

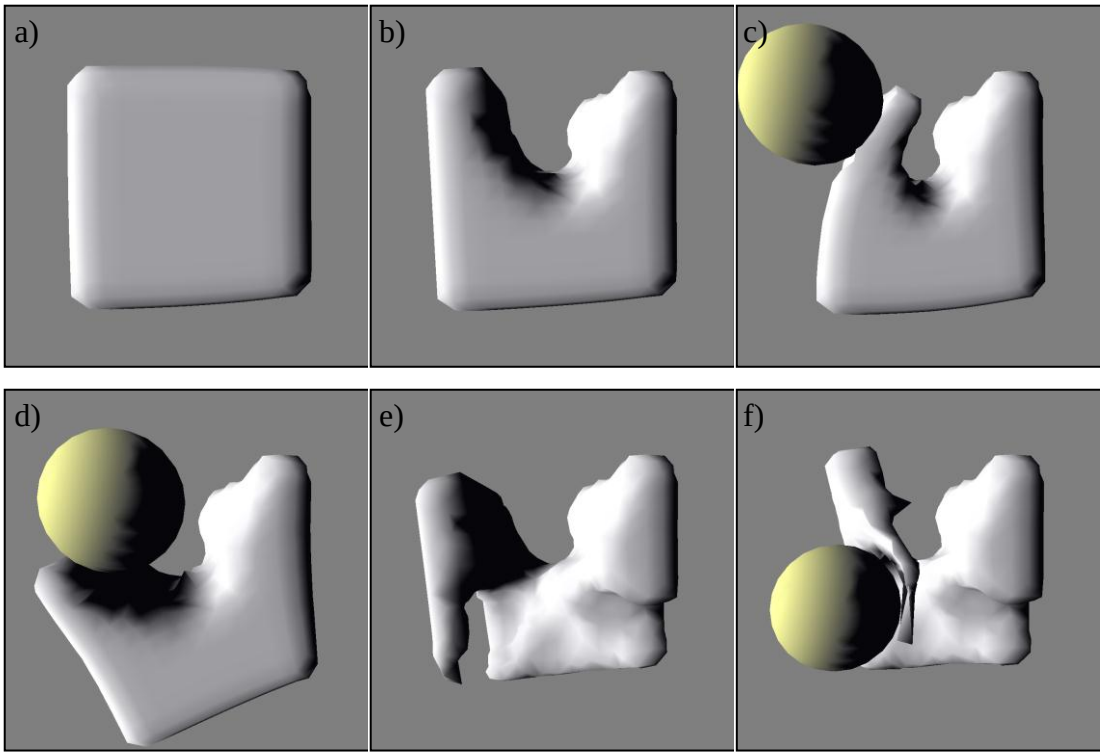


Figure 44: The tissue simulation being ablated and deformed interactively

The TSF can be tuned to deliver variable levels of mechanical fidelity or visual fidelity depending on the context of the simulation. Higher mechanical simulation resolution is useful for larger overall volumes and can also be used to retain a tighter correlation between the visual model and the mechanical simulation. The improved correlation between visual and mechanical models is due to the fact that the size of the volume corresponding to a set of eight mechanical simulation nodes defines the smallest volume which can be separated (mechanically) from the tissue

simulation. Therefore, a higher resolution mechanical simulation is required when fine details of the tissue must deform plausibly. Conversely, a coarse mechanical simulation can be used to reduce overall tissue simulation complexity (and GPU processing load) while retaining visual fidelity, provided that it is not important for finer details of the model to deform independently of larger volumes.

In addition to the versatility of the mechanical simulation, the TSF also offers a number of flexibilities that widen its capabilities and ensure that the requirements of other dependent systems can be accommodated. For example, haptic rendering requires a faster update rate than visual rendering. Since the TSF allows independent configuration of the mechanical simulation and the visual rendering, not only are developers able to trade-off visual fidelity for mechanical simulation fidelity, it also allows the processing load, and number of updates per render, of each sub-system to be managed separately. As a second example, the IMT component can be used with or without the optimisation or smoothing stages. Disabling either of these stages reduces the processing load during topology-changing interactions. Hence, if efficiency and high-rate updates are paramount then it may be better to disable these stages at the expense of decreased visual fidelity and a higher polygon count.

7.5 Summary

In summary, the CRMS mechanical simulation and IMT visualisation components have been successfully combined to produce a versatile tissue simulation with a range of new capabilities. The processing load of the CPU has been minimised by engineering a system that runs efficiently in parallel on the GPU. The relative resolution of each of the systems can be varied to allow greater emphasis on mechanical simulation or the render model.

Chapter 8. Haptics

If a picture is worth a thousand words, is a touch worth a million?

This chapter begins by briefly introducing the field of haptics and discussing the capabilities and shortcomings of the range of haptic hardware devices currently available. The body of the chapter details the methods used to add realistic tactile feedback (via haptic rendering) to the TSF (Aim 4) and critically discusses the quality of the haptic rendering achieved.

Haptic interfaces are those that use a tactile component to enhance user interaction. Broadly the term can include a large range of devices that deliver sensory input based on touch, including for example, mobile phones that vibrate in response to user input. In this thesis the term "haptics" refers only to precision force feedback as opposed to tactile displays, vibration-based effects or any other tactile stimulus modality.

The significance and importance of haptics to the efficacy of VR medical training simulation is a growing area of research. Haptics has a lot to offer computer-based medical training and indeed any training where precise manual skills are important.

Computer-based visual and auditory environments have matured to the degree that it now possible to deliver experiences in both modalities at levels of fidelity that approach the limits of human perception. In the case of computer-based visuals, colour accuracy, resolution, frame rates and lighting effects of computer generated imagery have become so realistic that it can be difficult to differentiate from reality. Increasingly high levels of visual realism are being

demonstrated in real time applications. However, haptic interactions have yet to reach the level where the sense of touch can be mimicked accurately.

Many medical interventions and therapies use the sense of touch to collect information and inform interactions. Using the appropriate level of force can be critical, particularly in delicate surgical procedures. Haptically enabled VR medical training simulation offers unique capabilities to improve training. However, the quality and reliability of the techniques used to deliver the haptic experience remain a limiting factor.

Haptic rendering is the process of computing and delivering haptic feedback forces. Speed is critical. While the human eye will interpret images that change at the rate of 30 to 60Hz as a continuous video stream, our sense of touch is far more sensitive in this respect and typically requires updates at the rate of at least 300Hz in order to deliver a stable and compelling haptic interaction [127]. This faster update rate together with other already relatively complex tasks, such as real time modelling of deformations and detecting collisions, makes development of reliable and compelling haptic rendering particularly challenging.

Figure 45: Immersion Corporation's LapVR

Figure 46: Simbionix's GI-Bronch Mentor

Haptics has effectively been used in a number of medical simulations, predominantly laparoscopic (Figure 45) or endoscopic (Figure 46) training systems [9, 10, 98, 100]. By limiting the number of degrees of freedom, and by using the same mechanical interface as the real instruments, these simulations effectively mimic the user interface of the instruments used to perform the real procedure.

Ideally, all simulations would accurately replicate the mechanical interfaces of the real instruments used. However, desktop haptic devices generally provide a generic handpiece that can represent the grip of a number of handheld instruments. Some instruments can be represented more effectively than others using this approach. Further, the lack of reactive torques in 3 degrees of feedback devices limits the types of instruments that can be effectively simulated.

8.1 6DOF Haptics

Desktop haptics devices have become cheaper in recent years. However, the price of haptic devices capable of rendering 6DOF force feedback (both linear forces and torque feedback) is still quite high. The cheapest 6DOF force-feedback haptic device from Sensable Technologies Inc. (USA) is upwards of AU$70,000 and competitors' prices are similar, although it is likely that in the future this competition will encourage price reductions and improved access to this important hardware.

For certain types of simulation, 6DOF feedback is critical. When an implement is grasped at a distance from the point of contact, any force not at the grip point, for example forces applied at the tool's tip, induces a torque (see Figure 47). A 3DOF feedback device is incapable of delivering such a torque, which makes creation of a compelling haptic interaction, where torques may at times be the dominant component of reactive forces, impossible. Hence, if a procedure makes significant use of tools that interact at a distance from the point the tool is held, then a 6DOF feedback device is required.

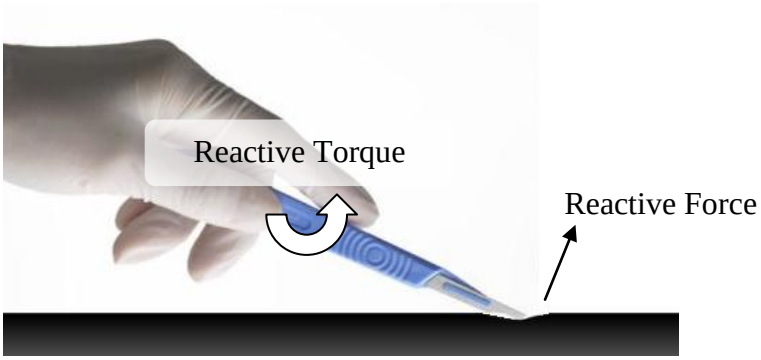


Figure 47: Contacts at a distance from the hand induce a reactive torque

The high cost of 6DOF feedback devices limits their use and instead motivates use of 3DOF feedback devices. One simple way of avoiding unrealistic interactions associated with the lack of reactive torque, is to move the centre of rotation to the tool's point of contact, often the tool's tip. This approach means that a 3DOF feedback device is sufficient, since rotating the tool does not move the point of contact. However, it will introduce a new, albeit lesser, problem since manipulating and rotating the tool will be somewhat strange and unintuitive since the user will now be effectively grasping the virtual tool's tip. If the resulting effect is distracting, or inhibits the use of the simulator, the use of a 6DOF feedback device may be unavoidable.

Figure 48: An endoscope connected to a 3DOFeedback haptic device at its tip delivers reactive torques to the user's hand

Finally, if a single tool is to be used, a 3DOF feedback device may be sufficient to provide a compelling interaction even for longer tools. The real tool can be modified and attached to the haptic stylus. Provided it does not mechanically interfere with the structure of the haptic device and allows unconstrained movement, the linear forces delivered to the tip of the tool are now experienced by the user as required, including the reactive torques (Figure 48). Other complications that may limit this approach include; weight of the tool/instrument, differences in the type of handle (for example scissor grips, that may require additional sensing

or force-feedback capabilities), and problems associated with excessive stylus inertia and friction.

8.2 Desktop Haptic Devices

Computer graphics and haptics are similar in that the quality of the user experience is determined by a combination of the software and the hardware. The visual experience depends both on the quality of the computer generated graphics, and also the quality of the hardware that displays it. Similarly, the capacity of the haptic hardware is a defining factor in the quality of the haptic interaction experience delivered by a haptically enabled simulation.

Compared to haptics devices, computer displays are a far more mature technology. When selecting hardware for a specific task it is relatively straightforward to find a display that is suitable for a given application, due at least partly to the standardisation of device specifications. Computer monitor capabilities are described in terms such as the resolution, contrast-ratio, maximum brightness, and pixel response times. These parameters accurately describe the device's capabilities, which enables device selection with a high degree of confidence that a given device will be fit for the intended purpose. On the other hand, specifications for haptic devices that are currently available can be difficult to interpret. With time these specifications will become more meaningful to those who use them, however, there are characteristics of haptics devices that are not included in existing specifications, such as what happens when the maximum force is exceeded. Table 2 lists the leading manufacturers of desktop haptic devices and the devices they offer. Figure 49 to Figure 54 show some of the most significant devices.

Figure 49 shows Sensable's entry-level desktop haptic device. The main limitation of this device is that it is limited to 3 degrees of force feedback. My own experience with this device has shown it to be capable of sufficiently accurate position (and orientation) input for most tasks (the level of precision of the sensors exceeds that which is perceptible to the user). The maximum feedback force it can deliver is 9N, which can be a limitation in some applications although it is ample for communicating surface boundaries to the user haptically.

Table 2: Leading desktop haptic devices by manufacturer

Manufacturer	Products	Sense	Feedback
Sensable	Phantom Omni	6DOF	3DOF
	Phantom Desktop	6DOF	3DOF
	Phantom Premium	6DOF	3DOF
	Phantom Premium 6DOF	6DOF	6DOF
Novint	Falcon	3DOF	3DOF
Force Dimension	Omega 3	3DOF	3DOF
	Omega 6	6DOF	3DOF
	Omega 7	7DOF	4DOF
	Delta 3	3DOF	3DOF
	Delta 6	6DOF	6DOF
Butterfly Haptics	Maglev 200	6DOF	6DOF
	Maglev 200 Grasp	7DOF	7DOF
Haption	Virtuose 6D Desktop	6DOF	6DOF
	Virtuose 6D35-45	6DOF	6DOF
	Virtuose 3D15-25	6DOF	3DOF
	Virtuose 6D40-40 (non desktop)	6DOF	6DOF
	Inca 6D (non desktop)	6DOF	6DOF

Figure 49: Sensable Phantom Omni

Figure 50: Novint Falcon

The other device I have experience with is the Novint Falcon (Figure 50). The specifications state that the maximum force it can deliver is also approximately 9N (given as 2lbs). However, using this device reveals that it is clearly capable of delivering much larger continuous forces. Passive movements of the stylus also require more force from the user to move it. Hence, this device is better suited to less precise, higher average force, applications.

Higher-end devices capable of deliver 6 degrees of force (linear and rotational forces) are also available (Figure 51 and Figure 52). Because of the high price of these devices I have not yet had access to them and cannot comment on their performance.

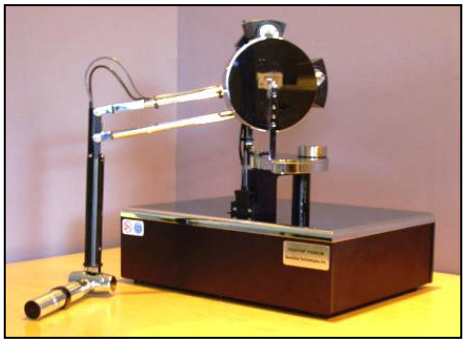


Figure 51: Sensable Phantom Premium 6DOF

Figure 52: Force Dimension Delta6

Butterfly Haptics provides a device that uses a different technology to deliver the force to the handpiece (see Figure 53 and Figure 54). Where other devices use stall-motors, this device uses direct magnetic coupling to deliver forces to the handpiece. This limits it to a small range of motion whilst allowing it to deliver accurate forces at very high rates.

Figure 53: Butterfly Haptics Maglev 200

Figure 54: Butterfly Haptics Workstation

All of the commonly available devices have small ranges of motion and therefore limited workspace. This restriction limits their relevance to simulations requiring large, free movements. However, since medical interventions commonly focus on relatively small areas of interest, these devices provide off-the-shelf solutions to what would otherwise be a very difficult capability to reproduce.

8.3 Haptics APIs

This section provides an overview of the currently available software tools that provide access to haptics devices. These are low-level interfaces simply for obtaining device state information such as position, orientation, button state and for setting the force vector to be delivered by the stylus.

The basic APIs from Sensable and Novint have similar capabilities; they provide interfaces to obtain position information (and in some cases rotational information) and button states together with the ability to control the feedback force delivered to the stylus. However, Sensable's APIs also provides higher-level functionality, which may simplify some development tasks relevant to medical simulation. In particular, it includes two notable options; the HL API and the Quick Haptics API. The HL API provides OpenGL application developers with a familiar method for haptically rendering 3D models. It is limited to rigid objects, although developers can specify when the model's geometry is modified such that the HL API may update the haptic render. Similarly, the Quick Haptics API includes support for haptic rendering that provides a system for very simple "mechanical simulation". However, the API supports local surface deformations only. For example, a horizontal structure anchored on one end will not deform, but the area around the point of contact will. The Quick Haptics API also provides a framework for the development of haptically enabled applications. However, since this API is provided by a commercial entity, anyone needing to alter the API or extend it to apply it to their application development or research will be encumbered by intellectual property issues.

CHAI 3D is "an open-source set of C++ libraries for computer haptics, visualisation and interactive real time simulation" [28]. Originally CHAI 3D's core component of interest was its device abstraction layer, which enables haptic devices from different manufacturers to use the same software interface. More recently, it has grown to include modules that support haptically interactive deformable bodies. CHAI 3D has been used to create a number of simulations, most recently a sinus surgery simulator [115].

H3DAPI is an open-source, cross-platform API for the development of haptically enabled applications. Like Sensable's Quick Haptics, it provides a framework for developing complete applications. It uses X3D (scene descriptions) and OpenGL for visual rendering. No direct support for deformable or cut-able tissue is provided.

Finally, Reachin Technologies is a company based in Sweden that has developed the Reachin API and the HaptX API. These APIs have a number of interesting capabilities, including networked haptics and deforming skin. However, since the API is commercial software (closed-source) it has limited relevance to this research. The available demonstration applications do not demonstrate, nor does the documentation show evidence of, support for cut-able models or global deformations.

8.4 Haptic Rendering

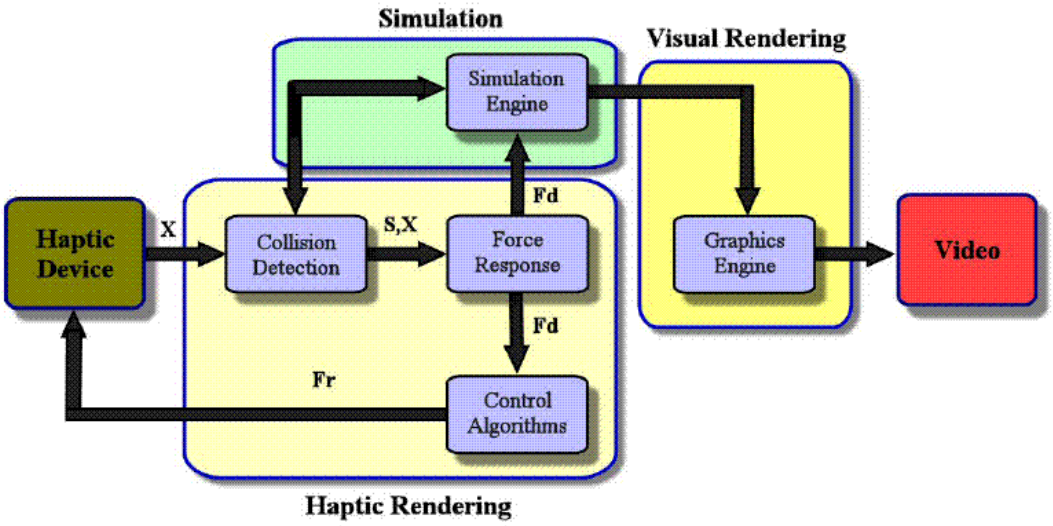


Figure 55: Block diagram of a haptically enabled simulation system [127]

In order to support haptic interaction a simulation must compute the reactive force that is delivered by the haptic device. However, the simulation must first determine whether the haptic stylus is touching an object using collision detection. Figure 55 illustrates a VR simulation system that includes a haptic rendering component. By separating haptic rendering components from the visual rendering, each sub-system can operate independently at their respective update rates (visual simulation updates of at least 30Hz, and haptic updates of at least 300Hz are required).

The reactive force should mimic the forces that the user's hand would experience when interacting with the real environment being simulated. That is, when the user moves the haptic device handpiece such that the haptic stylus inside the virtual environment touches an object, the simulation must compute the force that the stylus experiences as a result of that contact.

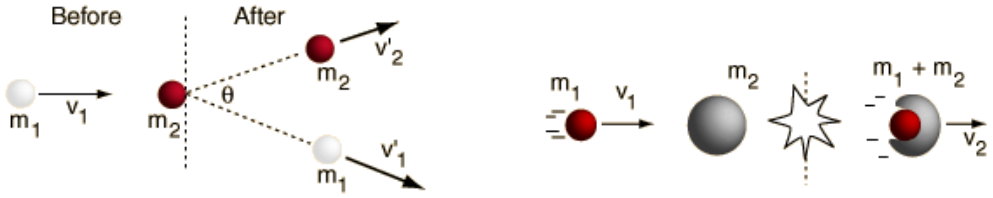


Figure 56: Elastic (left) and Inelastic (right) Collisions [104]

The forces that are sensed when a hand-held tool touches another object are the result of the two objects colliding. Hard, rigid objects result in a different experience to that of softer, and possibly globally deformable, objects. When hard objects collide the dominant forces are produced by an elastic collision. As the objects strike each other, objects exchange momentum and kinetic energy (Figure 56). Reactive forces occur in a direction perpendicular to the contact-surface, together with a tangential component for surface friction that resists motion (Figure 57).

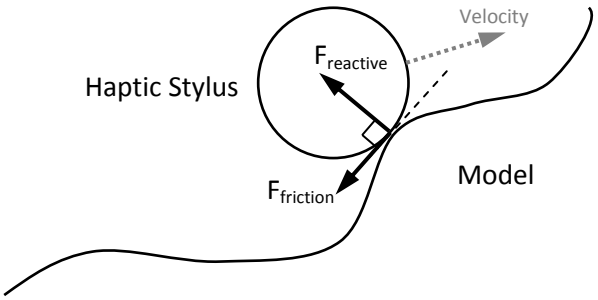


Figure 57: Haptic force components

Medical simulations in particular must simulate the collision between hand-held objects, such as surgical-instruments that are relatively light-weight when compared to the objects they touch. In such cases the tool tip will rebound from a hard surface with roughly the same kinetic energy as it had before colliding. Conversely, collisions with soft bodies are more complex; kinetic energy is lost, and the reactive force must be derived from the local state of the mechanical simulation.

As was the case with the mechanical simulation, the goal of achieving absolute accuracy is secondary to the goal of delivering a compelling user interaction. In addition, the force the simulation computes must be stable in situations that do not normally occur in the real world, such as when one object's model is inside another model. The simulation is also limited by the characteristics of the haptic device. Hence, haptic rendering requires new approaches that deliver a compelling and realistic user experience at interactive rates while minimising the use of processing and memory resources.

Different representations of 3-dimensional objects require different approaches. Models represented using a shell mesh rely on vertices and surface normals to compute collisions and subsequent reactive forces. Alternatively, in models represented volumetrically the intersection volume can be easily computed and used to compute reactive forces. The following sections detail how stable and compelling reactive forces can be computed from each model type (volumetric and shell-mesh). These algorithms have been used to add haptic capabilities to the TSF.

8.5 Haptics in the TSF

As described in previous chapters, the TSF represents the same 3-dimensional model in three ways, each of which provides a basis for the different capabilities of the system. A high-resolution volumetric dataset (Representation #1) is used to create the detailed render-mesh (Representation #2), that is warped by the deforming CRMS mechanical simulation (Representation #3) that is a cubic lattice. Depending on the application, each of these underlying representations can be used as the basis of the haptic rendering, and each has its own relative advantages.

8.5.1 Voxel-based Haptic Rendering

Volumetric representations of 3D models are amongst the simplest to compute reactive forces from, since testing for collisions between objects is as simple as testing whether any pair of voxels intersect. The challenge lies in performing collision detection and subsequent reactive force calculations (the basis of haptic rendering) with sufficient accuracy and rate that the user interaction is stable without requiring excessive processing.

The reactive force should always act in the direction that pushes the stylus away from the model so as to reduce the volume of intersection. With both the stylus and object models represented as volumetric models, the direction of the reactive force is computed from the centroid of the intersection volume and the centre of the spherical stylus model. (More generally, the centroid of the stylus rather than a sphere's centre can be used.) Further improvement to the accuracy, especially for soft touches, can be added by using the combined voxel densities of the intersection volume to weight the locations that are averaged to compute the centroid.

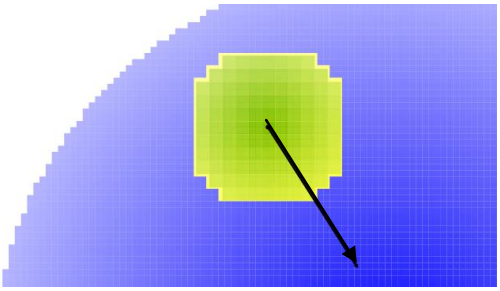


Figure 58: Reactive-force (black arrow) when models completely overlap

The calculation described above for the reactive force direction is only meaningful for partial intersections. The accuracy of the calculation decreases as the intersection volume passes the stylus centre. **Figure** 58 illustrates the reactive force direction that has been calculated using the described algorithm when the two models completely intersect. Hence, it is important to design the force-magnitude calculation such that the maximum magnitude for the reactive force is applied only when the error in the force-direction calculation is within acceptable tolerances. shows an example where the force direction is in the wrong direction due to the complete intersection of the stylus volume and the larger model (intersection centroid and stylus centre are coincident which means their delta cannot provide a meaningful direction).

Having shown that the valid range of the force direction calculation is limited, we must define a function for computing the force magnitude that has the desired characteristics for all cases. The force magnitude should increase smoothly as the intersection volume increases, then decrease smoothly as the maximum valid intersection point is reached. Further, the force magnitude should not contain any

discontinuities or excessively fast changes. If the force increases too rapidly in response to contact, soft-touches (small intersections) will result in "kick-back" when the delivered reactive force is momentarily high before the stylus moves so as to no longer intersect.

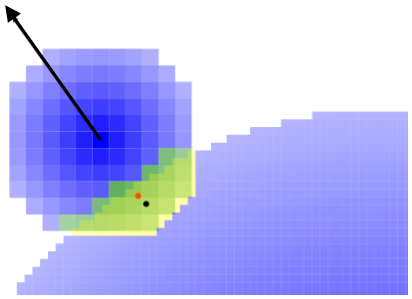


Figure 59: Coarse voxel sphere intersecting larger sphere

In order to conform to the previously defined specification the force is calculated according to **Equation** 12 below where V is volume.

$$F = (c_1 \mathrm{V}_{\mathrm{intersection}})^{c_2}, \text{where } \mathrm{V}_{\mathrm{intersection}} < 0.5 * \mathrm{V}_{\mathrm{stylus}}, \text{else } \mathrm{F} = 0$$

Equation 12: Force magnitude from intersecting voxel models

The force magnitude is simply the weighted volume of intersection (**Equation** 12). The shape of the force response curve (given by the above calculation across the range of valid intersections (i.e. intersections where less than half the stylus volume is intersected)) can be adjusted using the constants c_1 and c_2. The force magnitude is tuned such that the maximum force does not exceed the haptic device's maximum deliverable force, and c_2 is selected such that kick-back does not occur and small intersections are still perceptible.**Figure** 59 illustrates the force calculated using the described algorithm when a spherical stylus intersects a larger higher-resolution sphere. Two centroids of the intersection volume are marked; [red] is the centroid of the intersecting stylus voxels, [black] is the centroid of the intersecting model voxels. Voxels are not binary and rather can be thought of as containing a percentage occupied value. This is important for the tissue simulation since sets of eight voxels are used to compute the iso-surface (refer to Chapter 7). Using this algorithm for haptic rendering results in a smoother force curve (see section 8.6.2) than would be obtained using binary voxels.

Spatial partitioning can be used to increase the efficiency of this approach. However, since I have used direct addressing of voxels this will only assist with caching which only becomes useful when memory is saturated. In its described form, this approach is limited to the rigid (un-deforming) high-resolution representation of the model in the tissue simulation. However, the smaller model of the stylus can be inverse-warped according to the mechanical simulation state at the location of the stylus. This is equivalent to warping the high-resolution volumetric model of the tissue according to the mechanical simulation state and thereby achieves haptic-rendering of the deformed volumetric model.

8.5.2 Mechanical Simulation-based Haptic Rendering

Since the TSF mechanical simulation is structured as a cubic-lattice, it is equivalent to a coarse volumetric dataset. However, the critical difference here is that the system deforms and moves, which is especially significant once parts of the model are separated. In this situation, direct computation of the index of an intersecting node of the mechanical simulation is not possible.

The TSF uses coarse dynamic spatial partitioning to quickly detect which nodes intersect with the stylus. Nodes are grouped into axis-aligned bounding boxes (AABBs). Collisions between these boxes and the stylus are checked. If detected, the nodes within the box are checked individually for collisions with the stylus. Collision response moves the intersecting nodes to their nearest non-intersecting location. Reactive-forces are obtained directly from the mechanical simulation and summed for all colliding nodes.

Haptic rendering based on the mechanical simulation will only generate a convincing effect if the stylus intersects several nodes at once, otherwise moving the stylus across the surface of the model is experienced as bumps across individual nodes. Higher-resolution haptic rendering can be achieved using a voxel-based or surface-based method.

8.5.3 Isosurface-generated Mesh-based Haptic Rendering

Computing the haptic response from the IMT generated mesh is the most accurate, and the most processing intensive approach. Each of the previous approaches fails

to capture the maximum level of detail of the render model, which is only significant if the simulation relies on fine details of the surface to guide the interaction.

Using the previously described approach, nodes of the mechanical simulation near the stylus are computed. The association used to deform the mesh is then used to identify candidates for the narrow-phase collision test to identify individual intersecting triangles. This introduces problems where triangles of the mesh completely within the stylus mesh will not be part of the intersecting set. Hence, the stylus model uses a low-resolution volumetric model with stored associations to groups of surface triangles of the stylus collision mesh. In so doing, instead of tri-tri collision tests I use tri-sphere collision test (where each voxel is considered as an overlapping sphere). A weighted reactive force can then be calculated by weighting vertex normals in a similar way to that described for the volumetric haptic rendering.

8.6 Demonstration

Evaluations of haptic rendering algorithms are generally qualitative and subjective; unlike graphical rendering algorithms, the quality of the final effect cannot be easily shared because haptic effects cannot be communicated via screen-shots or video. Consequently new ways to concisely measure and communicate the subtleties of haptic rendering algorithms are required. Ultimately, it would be ideal if haptic rendering were evaluated in a standardised manner, using quantitative quality measures that completely describe the haptic experience in a manner that is compatible with any haptic rendering algorithm or device.

Ruffaldi *et al*. present a method for evaluating haptic rendering that uses "haptic trajectories" to capture the path taken by a haptic stylus for a small set of specific interactions [125]. They have made their data publicly available and encourage its use to compare other haptic render approaches with techniques they have used. Their method incorporates the use of precise position tracking and force transducers to completely describe the system, including the haptic device and the motion of the user's hand. Unfortunately the hardware used to track the haptic device and capture the actual forces is not readily available.

A key challenge of designing methods to test a haptic rendering technique is to constrain the variability to a manageable level without missing key characteristics of the haptic experience delivered. Ruffaldi *et al*'s method fully describes a small set of interactions for a limited number of operators (users). Factors such as the strength of the operator's grip on the stylus will affect the behaviour of the haptic device and consequently change the haptic trajectory which occurs. Other factors that significantly alter the haptic trajectory include the haptic device used, the biomechanics of the user's hand and arm during the interaction, and the grip used including the pose of the hand and the handpiece itself.

In order to provide a practical means to evaluating haptic rendering algorithms the variability of the test must be tightly controlled. The biomechanics of the user during testing is impossible to control completely, so let us remove it from the test system. Moreover, the test is further simplified by removing the haptic device altogether and using software to deliver a sequence of positions thereby moving the virtual stylus along a haptic trajectory. This creates a test system that can be completely defined, is readily reproducible, and can be captured accurately without specialised hardware.

8.6.1 Common Problems with Haptic Rendering Algorithms

During development of the TSF haptic rendering algorithms, a range of problems were encountered. This section defines the terminology used to describe the effectiveness of the haptic rendering of the tissue simulation.

8.6.1.1 Pop-through

Pop-through is the sudden decrease of reactive force that occurs when a collision test used to compute contact between the stylus and other colliding models suddenly fails. The problem is most likely to occur when haptic rendering is based on surface mesh models because of problems determining when the stylus model is completely contained within the surface mesh model.

Pop-through can also occur when the 3-dimensional models used to perform collision tests are shell meshes and the stylus model moves from completely external to too intersecting too far in a single time step. This can occur

if the frame-rate momentarily decreases, perhaps because of a momentary increase in CPU processing load. The stylus need only traverse the normal penetration depth between frames for pop-through to occur, since once intersections are too deep the haptic rendering algorithm may fail. (Continuous collision detection techniques [40] can be used to prevent pop-through.)

8.6.1.2 Jitter

Jitter is experienced as vibrations of approximately 10Hz to 100Hz when the stylus is in contact with a surface. It may be perceived as though the stylus is moving along a gritty or granular surface in situations where a smooth rendering is sought. The cause is usually inadequate haptic update rate. If the jitter frequency varies with the velocity of the stylus along the surface, the force calculation may be the cause, for example, if the force is computed from too few surface or volume elements.

8.6.1.3 Bounce

When grasping the haptic stylus with a firm but relaxed grip, bounce can be experienced in haptic rendering systems that are otherwise stable and compelling. Bounce is experienced as large amplitude (greater than 1cm) under-damped oscillations of the haptic stylus. The oscillation frequency may vary, but is typically under 10Hz.

Bounce occurs when the haptic update frequency is too low (under 300Hz). In mild cases the user can prevent this artefact by gripping the stylus more firmly or otherwise constraining movement of the stylus.

8.6.1.4 Kicks

Sudden changes to the force rendered to the haptic stylus are experienced by the user as “kicks”. These can be caused by transient errors in haptic rendering computations or sudden changes in force magnitude. Simply setting the scale of the haptic forces too high will result in a kick when a surface is softly touched.

Haptic rendering is a combination of delivering what the user expects and delivering accurate forces with a logical basis.

8.6.2 Voxel-based Haptic Rendering

The C++ implementation of the voxel-based haptic rendering algorithm described in section 8.5.1 is given in Code Listing 1. It is completely stateless; the result of the algorithm will be the same for any given position regardless of previous position and previous motion. Since stable haptics requires an update rate of at least 300Hz it is important to minimise the computations required to calculate the haptic forces. An obvious way to do this is to minimise the resolution of the voxel datasets used. However, as shown in Figure 61 and Figure 62, when the voxel models used consist of a low number of voxels (in this case the stylus model is 8^3 voxels and the sphere model 32^3 voxels (see Figure 60)), the computed force is plausible, but not smooth. This is experienced by the user (via the haptic stylus) as jitter (as described in section 8.6.1.2).

Code Listing 1: Computing the Voxel-based Reactive Force

```
1   vec3f ComputeHapticForce()
2   {
3       vec3f force(0.0f, 0.0f, 0.0f);
4       vec3f centroid(0.0f, 0.0f, 0.0f);
5       float total = 0.0f;
6       vector<float>::const_iterator voxel = collVoxels.begin();
7         for (vector<vec3f>::const_iterator pos =
8              collVoxPositions.begin();
9              pos != collVoxPositions.end(); ++pos, ++voxel)
10      {
11          centroid += ((*pos) * *voxel); // weighted centroid
12          total += *voxel;
13      }
14      centroid /= total;
15
16      if (collVoxPositions.size() < halfStylusVoxCount)
17      {
18        force = normalize(stylusCentre - centroid) *
19                pow(c1 * total, c2);
20      }
21      return force;
22  }
```

Looking at Figure 61 and Figure 62 in more detail: As the two spheres touch, the volume of intersection is small with its centroid a maximum distance from the stylus centre. The maximum force occurs at $0.5*r_{stylus}$. The chart has been normalised such that the two maxima have a force of one. The force is scaled in the tissue simulation such that the maximum force set by the software is close to the maximum supported by the haptic device used. If the force is scaled too high kicks will occur. Figure 62 shows the result of the dot product of the force vector and the

direction of motion. Initially there is no contact so the direction is not computed. Once the two spheres touch the direction computed force is applied in the direction that opposes the intersection of the volumes. As the spheres intersect further the direction eventually switches as the stylus moves past the centre of the model. Notice that while the stylus is near the centre of the model there is variation (error) in the direction caused by inaccuracies of the rendering algorithm which uses the distance between the centroid of intersection and the stylus centre which are coincident.

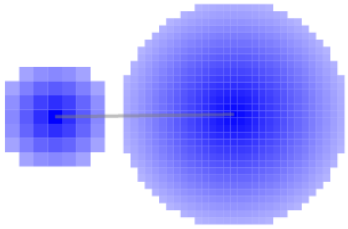


Figure 60: Coarse sphere-sphere intersection test

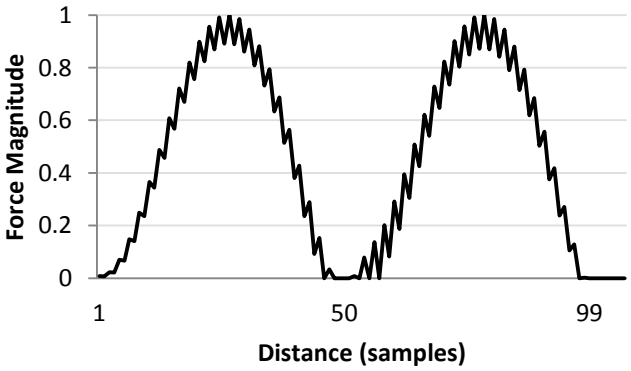


Figure 61: Force magnitude (coarse spheres)

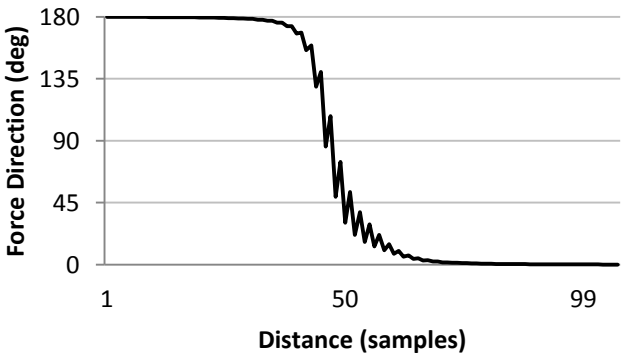


Figure 62: Force direction (coarse spheres)

Increasing the resolution of the voxel models used increases the level of detail they provide. As shown in Figure 63 and Figure 64 the haptic forces computed are free of

jitter artefact. However, the computations required to update haptic forces via the method previously described increases linearly with n (where n is the number of voxels).

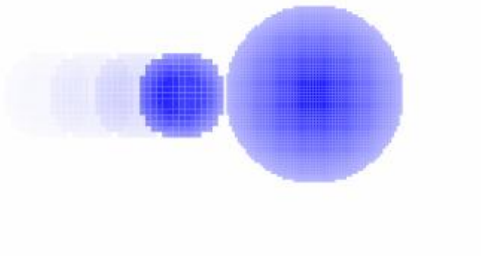


Figure 63: Fine spheres

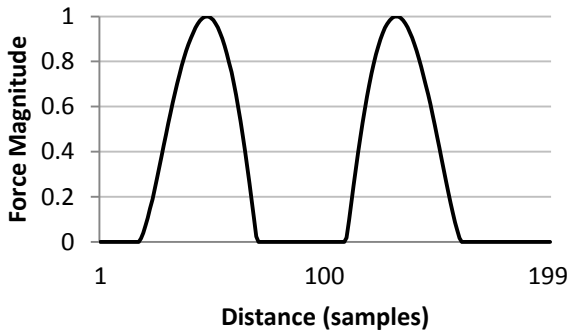


Figure 64: Force magnitude (fine spheres)

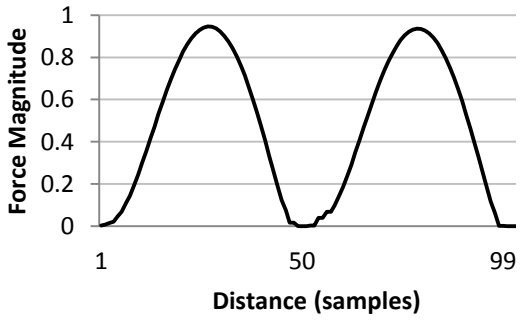


Figure 65: Smoothed force magnitude (coarse spheres)

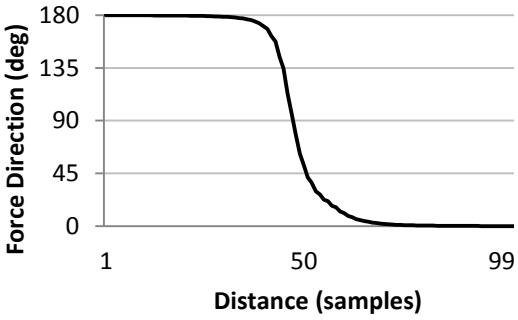


Figure 66: Smoothed force direction (coarse spheres)

One option that has proven an effective means to reduce jitter for lower-resolution voxel model-based haptic rendering is to use a simple averaging window to smooth the output. Figure 65 and Figure 66 show the result of applying a 2-sample wide averaging window to the data in Figure 61 and Figure 62; the jitter has been removed. Use of the averaging window introduces latency into the system. However, provided the introduced delay does not exceed a few milliseconds, jitter can be removed without introducing bounce (see section 8.6.1.3). This is one way in which increased latency can be traded for reduced computational load. Other methods for reducing computational load for haptic rendering of higher resolution voxel models include hierarchical data structures.

A final example of the behaviour of the haptic rendering algorithm is shown in Figure 67 where the stylus is moved though a chamfered cube. Figure 68 shows the output of the haptic rendering algorithm as two smooth peaks. The zero region between these peaks demonstrates that the algorithm is behaving correctly by masking the force when the two models fully intersect (to avoid jitter and more severe artefacts caused when the centroid of the intersection volume is coincident with the stylus centre). Figure 69 shows that the correct direction has been computed; 180° initially as the reactive force acts in the opposite direction to the motion, and -135° as the stylus exits the far side of the model.

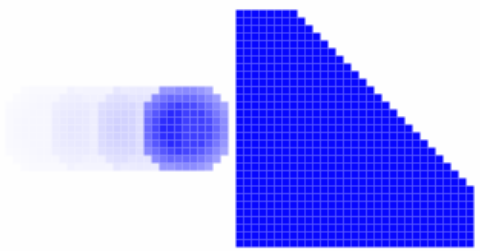


Figure 67: Sphere passing through a chamfered cube

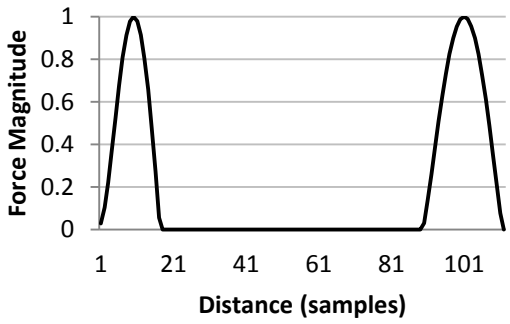


Figure 68: Force magnitude (sphere through chamfered cube)

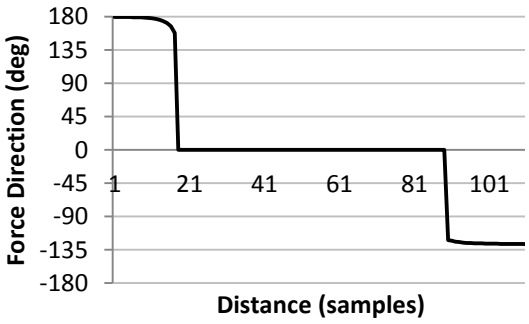


Figure 69: Force magnitude (sphere through chamferd cube)

These examples demonstrate that the voxel-based haptic rendering algorithm used in the tissue simulation is reliable and delivers a plausible user experience. However, appropriate model resolutions must be selected to avoid performance problems. A simple averaging window is a useful means for removing jitter and maintaining a compelling user interaction. However, since the averaging window introduces latency it cannot be used in all cases without causing haptic rendering artefacts such as bounce.

8.6.3 Mechanical Simulation-based Haptic Rendering

Voxel-based haptic rendering allows models to intersect and move through each other unlike the haptic rendering based on the mechanical simulation. Here a spherical stylus model is moved vertically downward into a mechanical simulation of a beam anchored on the right side (Figure 70). The beam is relatively soft and deforms as the stylus model is moved downward.

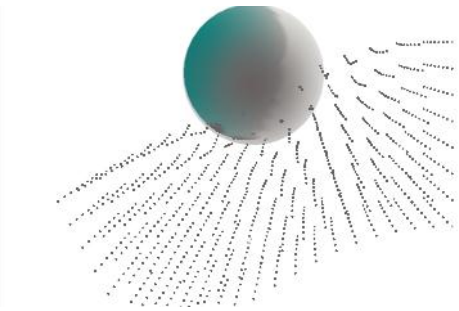


Figure 70: Spherical stylus interacting with CRMS mechanical simulation

This haptic rendering approach is not based on an intersection volume (as is the voxel-based approach). Instead the stylus model displaces nodes of the mechanical simulation and the reactive forces generated by the mechanical simulation are accumulated and applied to the haptic stylus (refer to section 8.5.2).

As the stylus moves down and collides with the mechanical simulation model, the colliding mechanical simulation nodes are moved implicitly at the same velocity as the stylus. Figure 71 and Figure 72 show the haptic rendering output when the stylus is moved. The slight decrease in the angle shown in Figure 72 is a result of the deformation of the model and the consequent rotation of the computed reactive force. Notice that the force delivered with this method is quite rough (red line). Introducing a short (4-sample) averaging window improves this but the rendered force is still not perfectly smooth (blue line). The reason for the roughness is that individual nodes have mass and therefore inertia, which causes them to oscillate after they are displaced (due to the collision with the stylus). The collision response does not use common collision principles at all (no explicit momentum or kinetic energy transfer are used) and instead sets the node velocity to zero after it is displaced to minimise the "bounce" effect. The apparent roughness in the force calculation does not affect interaction with the model.

The small variations in the haptic rendering output are not significant and the final effect is compelling and reliable. Moreover, the algorithm is efficient and works well with the complete tissue simulation.

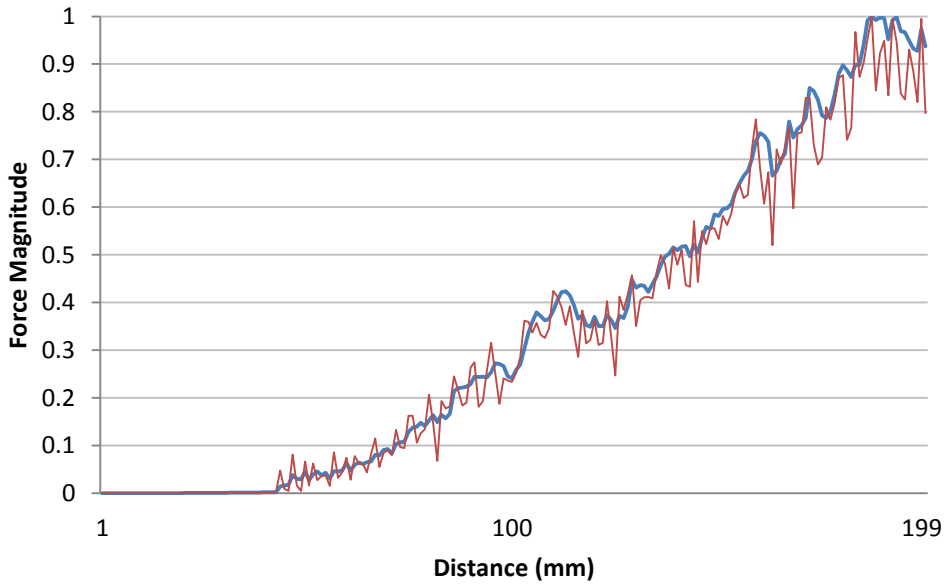


Figure 71: Force Magnitude (CRMS)

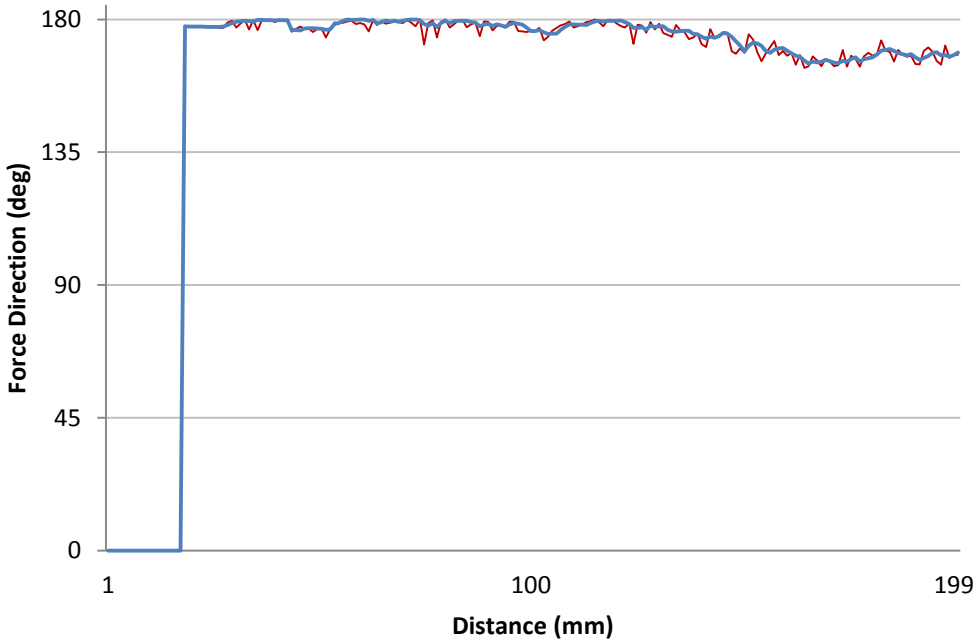


Figure 72: Force direction relative to stylus motion (CRMS)

8.7 Summary

As access to 6DOF-feedback haptic devices improves, haptics will continue to develop in its ability to enhance simulations. Medical training simulations in

particular stand to gain a lot from this relatively new capability, particularly where manual dexterity or the forces felt through surgical instruments informs interaction.

A small number of device manufacturers exist who are producing different types of haptic devices. There are software libraries available that provide varying levels of functionality. Some of these go beyond simple interfaces to manage the device state while others provide support for haptic rendering. In the end, the haptic algorithm used to compute reactive forces determines the fidelity of the haptic user experience. Despite these libraries, there are still opportunities to enhance the haptic experience a particular system can provide by developing haptic rendering algorithms which work natively with the model formats available at run time.

This chapter has described three haptic rendering approaches which work directly with the different model representations present in the TSF. If it is important that high-resolution details of the volumetric model be rendered haptically, then a voxel-based can be used. If speed and computational efficiency are critical, then the mechanical simulation-based haptic rendering algorithm described is also capable of delivering stable and compelling force feedback for haptically enabled VR medical simulations.

Chapter 9. Applications

This chapter demonstrates the success of the TSF by describing several VR medical simulators that have been developed using the tissue simulation to enable higher quality key interactions.

9.1 An Endoscopic Sinus Surgery Simulator

Endoscopic Sinus Surgery (ESS) is a relatively modern minimally invasive approach to sinus surgery, the popularisation of which has been credited (by Wormald [162]) to Stammberger [141] and Kennedy [73] in the mid 1980's. This surgery is performed on patients under general anaesthetic. During the surgery an endoscopic camera is used to view the sinuses while instruments are used to remove and manipulate tissues within the sinus cavity (Figure 73 - Figure 75).

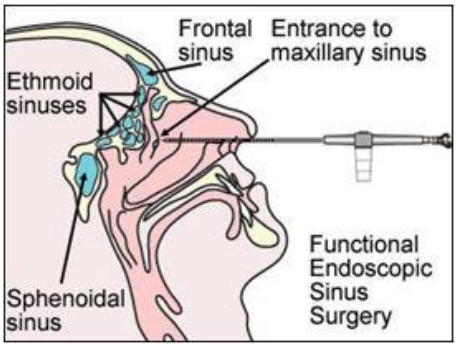


Figure 73: Cross-sectional view of ESS [124]

Figure 74: Patient undergoing ESS [162]

Because of the close proximity of the sinuses to the orbits (eye sockets) and the brain, the surgeon must be especially careful; the bone separating the sinuses from the brain and orbits is quite delicate. Consequently, all ESS procedures carry a small risk of causing eye problems or spinal fluid leaks (which in turn can cause meningitis) [124]. The extra opportunities for surgical training provided by

simulation are therefore a promising avenue for reducing the frequency of complications and maximising patient safety.

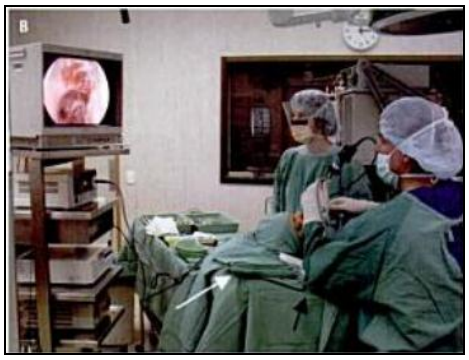


Figure 75: Operating room setup for ESS [162]

Figure 76: Paediatric backbiter reference images

Despite some prior work developing ESS simulators [39, 115], the expense and lack of fidelity have limited access to these simulators and their effectiveness. The TSF provides key functionality that can address limitations of the existing simulators. Multiple instances of the tissue simulation are integrated into the simulation in order to allow users freedom to modify key anatomical structures within the sinuses using a range of surgical instruments (such as paediatric backbiters Figure 76 - Figure 78).

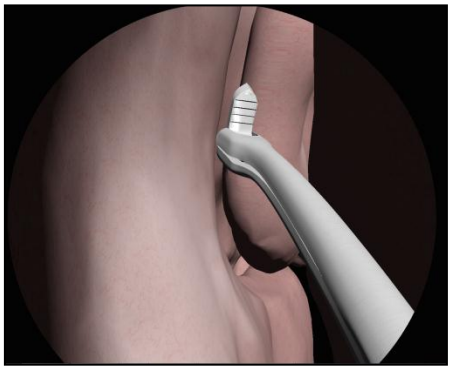


Figure 77: The sinus simulator

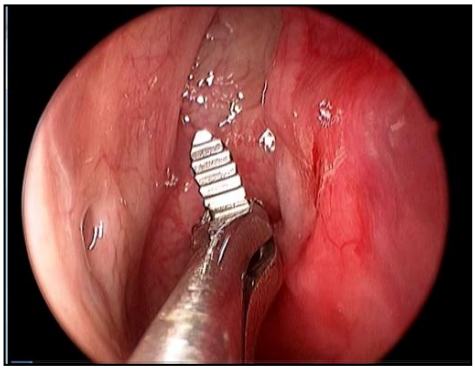


Figure 78: View of the sinus during surgery

At the time of writing, development has resulted in a prototype (see Figure 77 and Figure 80) that supports two-handed interaction (one controls the endoscopic camera, the other controls a surgical instrument). The prototype uses anatomically accurate 3-dimensional models generated from patient CT data. Further, the models are processed to replicate the contraction of the soft sinus tissues that results from the administration of vasoconstriction inducing drugs, such as adrenaline spray, as is normally present during sinus surgery [119]. This effect is

simulated by using a vertex shader to offset vertices in the direction of the vertex normal. Figure 79 shows a view of the sinus model (the middle structure is the middle turbinate) where the amount of tissue contraction is varied from a maximum (left), to the original model (right).

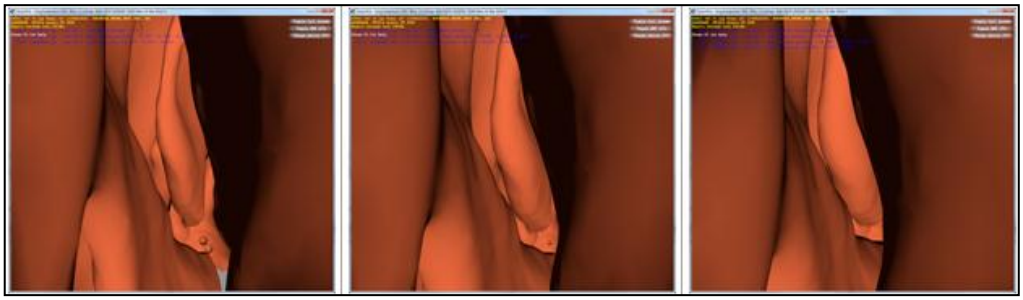


Figure 79: Sinus soft-tissue contraction (left: contracted, to right: original model)

Work on this project is continuing with funding from The Garnett Passe and Rodney Williams Memorial Foundation. The project aims to provide a prototype simulator to each of the five states of Australia with major ENT training rotations by early 2013. The TSF provides important functionality required to simulate the uncinate process, amongst other structures, that are modified or removed during simulated surgical procedures.

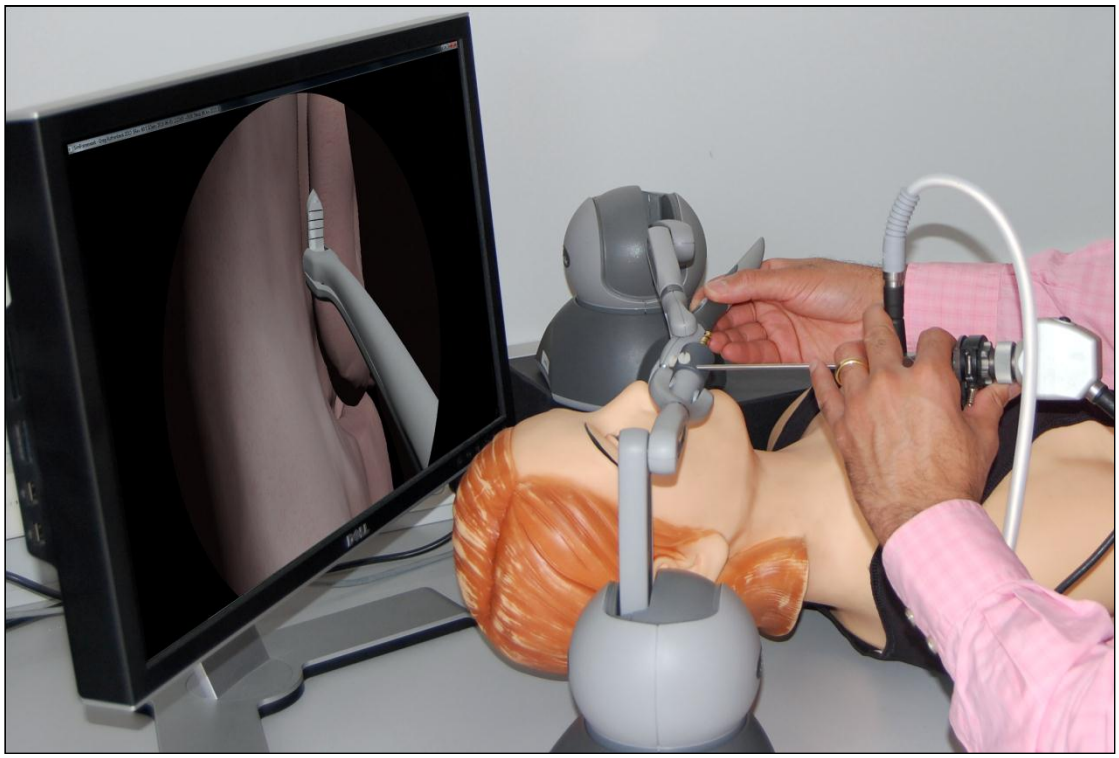


Figure 80: The sinus simulator in use

9.2 ISim: An Endotracheal Intubation Simulator

Endotracheal intubation is a difficult procedure commonly performed as part of delivering a general anaesthetic or when maintaining a clear airway is otherwise difficult. Students typically gain experience performing the procedure on real patients, which both presents a risk to patients and also limits students' opportunities to master the skills required. Haptically enabled VR simulation presents new opportunities to develop skills trainers for practicing endotracheal intubation on a range of virtual patients without exposing real patients to unnecessary risks.

The process of endotracheal intubation begins by raising and tilting back the head (Figure 81). Then the right hand is used to open the mouth and carefully insert the laryngoscope blade into the mouth (Figure 82). The laryngoscope blade is then used to control the tongue lift it to obtain a clear view of the epiglottis. The blade tip is then positioned between the base of the tongue and the valecula (above the epiglottis) where pressure is applied to open the epiglottis and obtain a clear view of the vocal chords. Finally, a tracheal tube is inserted through the vocal chords into the trachea where a cuff around the tube tip is inflated to achieve an air-tight seal.

Figure 81: Performing Intubation (with assistance)

Figure 82: A laryngoscope

Figure 83: Laryngoscope handle attached to haptic device

An endotracheal intubation simulator, ISim, was implemented using the TSF (Figure 84 and Figure 85). It builds upon a previous version (detailed in Appendix B) by using the TSF to improve the realism of the behaviour of the tongue, and the quality of the tactile feedback. The virtual laryngoscope is controlled via a Sensable Phantom Omni with a modified standard laryngoscope handle attached

(Figure 83). The TSF has improved the key interaction and is expected to result in improved learning outcomes.

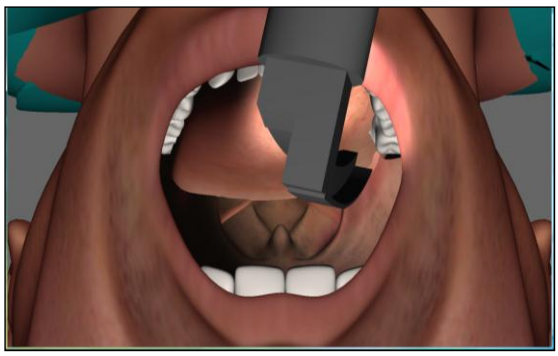


Figure 84: ISim screenshot

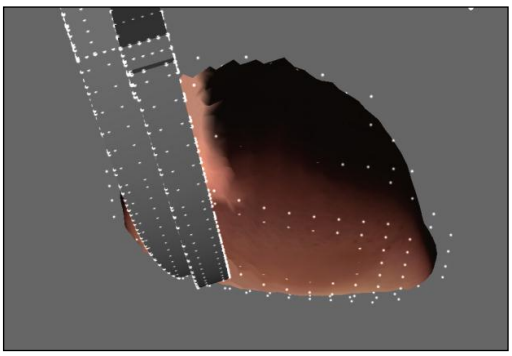


Figure 85: TSF-based deforming tongue model

Work on this simulator is continuing with funding from the Australian Research Council. A commercial partner has been engaged to continue development of this simulator and pursue commercial opportunities for this simulator.

9.3 A Coblation Tonsillectomy Simulator

Although the number of tonsillectomies performed annually is estimated at half of the rate 40 years ago [164], tonsillectomy remains a very common surgical procedure. According to Paradise *et al*. [114] "tonsillectomy is the most commonly performed major surgical operation among United States children". In a recent survey taken in 1996, some 287,000 children under 15 years of age underwent tonsillectomy in the US alone [112]. This is supported by statistics for the UK which put the incidence of recurrent sore throat in general practice at 10% [113].

Fortunately surgical practices have developed since Cornelius Celsus of Rome first described how to perform a tonsillectomy around 40 A.D. [47]. New tonsillectomy surgery techniques and technology reduce risks and generally improve outcomes. Coblation is a relatively new method for performing tonsillectomy that "is associated with less postoperative pain and early return to daily activities. Also, there are fewer secondary infections of the tonsil bed and significantly lower rates of secondary haemorrhage with coblation" [13]. Other research supports these claims [26, 37, 93, 117, 143, 150, 151], although Noon *et al*. [107] report "significant increase in the secondary haemorrhage rate". Overall,

the literature supports coblation as a modern technique that results in faster recovery and reduced complications.

Coblation in this context is literally the removal of tissue using low-frequency RF ablation. Rather than cut tissue, coblation causes molecular dissociation at a cellular level [16]. Consequently the surgical technique required differs significantly from conventional surgery.

The surgical technique used to perform tonsillectomy "consists of dissection in the subcapsular plane" [70] to remove the tonsils and adenoid. Traditionally this dissection is performed using a scalpel and optionally forceps to carefully cut or tear the tonsil away. In contrast using the coblation handpiece (Figure 87) requires "feather pressure" to slowly separate the tonsils from the substrate. Learning the correct pressure and technique is critical.

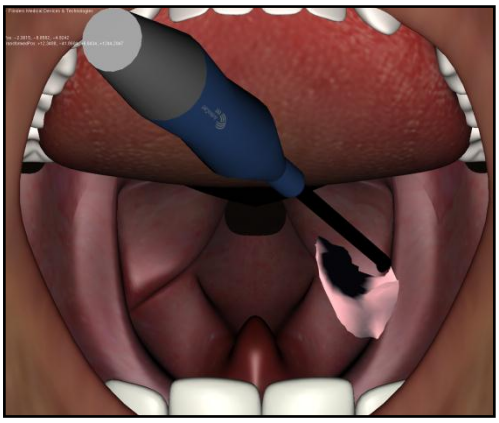


Figure 86: Coblation simulator screenshot

Figure 87: Coblation Handpiece

A coblation tonsillectomy simulator has been developed using TSF together with expert practitioners from the Flinders Medical Centre's otorhinolaryngology department with funding from a significant North American Medical Device manufacturer (Figure 86). The simulation was written in C++ using NVSG (refer to section 3.1.1). The TSF provides critical functionality to allow the interaction of the coblation handpiece to be simulated in a realistic manner. Development and feedback from clinicians is expected to result in successful publication of this work later this year.

9.4 Summary

The TSF has served as a key component of a number of simulations. Its versatility, though not fully exercised in these applications, has improved the key interaction of these simulations. Moreover, as the code base that implements the tissue simulation matures it will facilitate development of new types of simulations that would otherwise have been very difficult, or impossible, to create.

Chapter 10. Conclusion

The TSF provides an efficient new method for simulating a interactive tissue for VR medical simulation applications. The CRMS mechanical simulation component incorporates several innovations that enable it to plausibly simulate a broad range of mechanical properties not normally associated with mass-spring based methods. The system incorporates performance optimisations that play to the strengths of GPU hardware, such as memory usage patterns that allow coalesced memory access. The IMT surface-generating component builds upon the success of existing refined marching algorithms to enable high-resolution visualisations at unprecedented interactive rates. The key innovation is reducing the processing cost of updates by updating only small sub-volumes that have changed. The data output by the IMT component is formatted for compatibility with typical geometry streams to enable rendering with a multitude of shading techniques. Additional visual fidelity is added using optional mesh optimisation and vertex-normal smoothing stages. The combined components produce a much needed reusable software component capable of adding critical functionality to a broad range of VR medical simulation applications. To enhance the simulation experience, haptic rendering methods were developed that allow much needed tactile feedback to be based upon the CRMS mechanical simulation directly, or alternatively the higher resolution IMT volumetric model. A number of VR medical training applications that demonstrate the utility of the TSF have been developed.

10.1 Future Directions

The TSF can be enhanced in a number of ways to aide continuing development of improved medical simulations (including those described in Chapter 9):

10.1.1 Material Library

The applications developed using the TSF (Chapter 9) have employed manual tuning of mechanical simulation parameters to achieve the required mechanical characteristics. It would be useful to develop a material library of commonly used tissue types to simplify re-use and the development of other simulations in the future. This could be based on experimental data (similar to [99]), however it should be sufficient to develop preset materials and refine them based on feedback from expert clinicians.

10.1.2 Rapid Prototyping of Patient-Specific Simulations

A material library, together with tools for importing patient data would enable the TSF to be used to rapidly prototype a large range of patient-specific medical simulations. Such a tool would be a valuable asset to VR-based medical training simulation developers and researchers. With sufficient refinement and development it would be an ideal way to empower clinicians and medical trainees to contribute to the development of their own simulations based on this technology.

10.1.3 Performance Optimisations

10.1.3.1 Adaptive Tesselation of the CRMS Lattice

Adaptive tesselation of the CRMS lattice would enable sets of nodes within the mechanical simulation to be grouped and replaced with simpler elements. However, this approach is difficult to achieve without significantly reducing the efficiency of the current system since it removes the ability to perform direct implicit node addressing, which is important for exploiting memory coalescing and efficient parallelisation.

10.1.3.2 Increase the Maximum Resolution of the Mechanical Simulation

The existing implementation of the CRMS mechanical simulation (described in Chapter 5) is limited by the maximum number of simultaneous threads that can be executed in parallel on the Nvidia GTX 280. A tiling mechanism could be used to spawn sets of threads. This would increase the maximum resolution supported by

the system since the existing system performance is not processing or memory limited.

10.1.3.3 Smart Node Update Scheduling in the Mechanical Simulation

The regular cubic lattice used by CRMS can result in significant internal volumes where the state of the system remains relatively constant. It may be possible to improve performance by reducing the frequency of updates to nodes that aren't undergoing significant motion. Again, the challenge with this approach is efficiently implementing it in parallel. One important mechanism could be to use a pre-sort to identify the subset of nodes requiring an update. If this can be done using less processing than the processing of updates at a uniform frequency, then overall performance will be improved.

10.1.3.4 Per-Spring Cutting

A tighter correlation between the high-resolution visible model and the mechanical simulation could be achieved by softening, and breaking, individual spring connections, rather than removing nodes from the mechanical simulation. This will introduce additional processing and memory overheads. Whether it can be achieved without excessive overheads is the subject of continuing research.

10.1.3.5 Exploit Texture Memory and Raster-Operations

Higher performance may be possible by making more extensive use of texture memory and exploiting the hardware capabilities dedicated to raster-operations (ROPs). Even current general purpose graphics processors have considerable additional processing capabilities dedicated to tasks such as texture sampling and texture mixing. These capabilities are referred to as raster-operations (ROPs).

10.1.4 Volumetric Overlays

When tissues are cut during surgery there are often anatomical details, such as blood vessels and nerves, that should be avoided. Often these obstacles exhibit different properties. Cutting or ablating them may also cause responses such as bleeding. Moreover, some surgeries target these structures and hence the simulation requires them to have higher fidelity than the tissue substrate.

Since the TSF employs a high-resolution volumetric model, these features could be added simply by defining a reserved range of densities (whereby small density ranges represent different tissues such as blood vessels), or alternatively, using a separate volumetric dataset to describe their location within the tissue. Certain values in the high-resolution volumetric model, once exposed by cuts, could trigger bleeding or other secondary effects. Moreover, since IMT can create a surface that follows any arbitrary iso-value, dual surfaces can be generated and maintained from a single dataset (just as normally occurs when segmenting CT data) whereby a main surface has additional detail added by secondary iso-surfaces that represent other tissue types within the same model.

Finally, the surgical instrument can be made to behave differently with the various tissue types to simulate the way real tissues behave. For example, when a scalpel is gently run through soft fatty tissue, the fatty tissue is easily cut. Cartilage requires more pressure to cut. This can be added to the TSF by changing the rate at which the cut progresses depending on the pressure of the blade. When implemented at sufficient resolution, this effect would greatly enhance the realism of certain types of interactions central to many surgical procedures.

10.1.5 Haptic Render Testing and Evaluation Using a Psycho-motor Model of the Hand and Arm

The haptic experience is influenced by the biomechanics of the user and their grip on the haptic stylus. A strong grip and heavy hand will produce a different haptic experience than someone gripping the stylus lightly. The haptic experience is also changed by the user's expectations as the stylus is moved. For example, if the user is striking a hard object to shatter it, the user will intuitively grip the stylus more firmly as the surface is struck. Hence, haptic rendering algorithms must behave reliably when mechanically coupled to (held by) differing biomechanical systems (hand and arm). In summary, when the user expects a certain force response, their grip changes, which in turn impacts the haptic rendering system by changing the haptic trajectory.

The term ragdoll is used to describe the kinematic model of a passively moving human model. Ragdolls have been used in video games to model a falling

dead body, for example in the PC game series Hitman (developed by IO Interactive). More complex ragdoll like systems have been developed to reduce the cost of developing content for video games and movies (for example, Euphoria developed by Natural Motion Ltd. [1]). Of particular relevance is the work of Natural Motion Inc who have products that allow for intelligent ragdoll animation creation. Euphoria incorporates a model of the human nervous system to mimic the reflexes and responses to certain stimuli. For example, a surprised player in a rugby tackle will been more lucid, a ready player will be braced, and the subsequent simulation will consequently proceed very differently. A similar type of model of the user's hand and arm would enable more comprehensive automated testing of haptic rendering algorithms and perhaps even standardised testing that could be used to compare haptic rendering algorithms in a more controlled manner.

10.2 Final Words

The TSF fulfils an important need that had been missing from the tools available to developers of VR medical simulations. By efficiently leveraging modern parallel computing hardware it provides a realistic and versatile interactive tissue simulation that can be cut and ablated without restriction. Its ongoing development and use in a range of VR medical simulations such as those described in Chapter 9 has the potential to revolutionise the training of medical practitioners.

Bibliography

1. *Natural Motion Euphoria*. [cited 2010 May 20]; Available from: http://www.naturalmotion.com/euphoria.htm.
2. *Id Software*. 2010 [cited 2010 15 May]; Available from: http://en.wikipedia.org/wiki/Id_Software.
3. Abrahamson, S., Denson, J., and Wolf, R., *Effectiveness of a Simulator in Training Anesthesiology Residents*. 2004, Quality and Safety in Health Care. p. 395-397.
4. Allard, J., Cotin, S., Faure, F., et al., *SOFA - an Open Source Framework for Medical Simulation*, in *Medicine Meets Virtual Reality 15*. 2007, IOS Press: Long Beach, CA. p. 13-18.
5. Barach, P. and Johnson, J.K., *Reducing Variation in Adverse Events During the Academic Year.* British Medical Journal, 2009. **339**(1): p. 3949.
6. Barbe, W. and Milone Jr, M., *What We Know about Modality Strengths.* Educational Leadership, 1981. **38**(5): p. 378-80.
7. Barber, C., Dobkin, D., and Huhdanpaa, H., *The Quickhull Algorithm for Convex Hulls.* ACM Transactions on Mathematical Software, 1996. **22**(4): p. 469-483.
8. Barry-Issenberg, S., McGaghie, W., Petrusa, E., et al., *Features and Uses of High-Fidelity Medical Simulations that Lead to Effective Learning: A BEME Systematic Review.* Medical Teacher, 2005. **27**(1): p. 10-28.
9. Basdogan, C., Ho, C., and Srinivasan, M., *Virtual Environments for Medical Training: Graphical and Haptic Simulation of Laparoscopic Common Bile Duct Exploration.* IEEE/ASME Transactions on Mechatronics, 2001. **6**(3): p. 269.
10. Basdogan, C., De, S., Kim, J., et al., *Haptics in Minimally Invasive Surgical Simulation and Training.* IEEE Computer Graphics Applications, 2004. **24**(2): p. 56-64.
11. Bates, D.O., Levick, J.R., and Mortimer, P.S., *Quantification of Rate and Depth of Pitting in Human Edema Using an Electronic Tonometer.* Lymphology, 1994. **27**(4): p. 159-172.
12. Baykan, Z. and Nacar, M., *Learning Styles of First-Year Medical Students Attending Erciyes University in Kayseri, Turkey.* Advances in Physiology Education, 2007. **31**(2): p. 158.

13. Belloso, A., Chidambaram, A., Morar, P., et al., *Coblation Tonsillectomy versus Dissection Tonsillectomy: Postoperative Hemorrhage.* The Laryngoscope, 2003. **113**(11): p. 2010-2013.

14. Benthin, C., Wald, I., Scherbaum, M., et al. *Ray Tracing on the Cell Processor*. in *IEEE Symposium on Interactive Ray Tracing*. 2006. Salt Lake City, UT.

15. Berkley, J., Turkiyyah, G., Berg, D., et al., *Real-Time Finite Element Modeling for Surgery Simulation: An Application to Virtual Suturing.* IEEE Transactions on Visualization and Computer Graphics, 2004. **10**(3): p. 314-325.

16. Bortnick, D., *Coblation: An Emerging Technology and New Technique for Soft-Tissue Surgery.* Plastic and Reconstructive Surgery, 2001. **107**(2): p. 614.

17. Bourke, P. *Polygonising a Scalar Field (Marching Cubes)*. 1994 [cited 2010 5-May]; Available from: http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/.

18. Boyd, C. *DirectX 11 Compute Shader*. in *SIGGRAPH*. 2008. California, USA.

19. Branch Jr, W. and Paranjape, A., *Feedback and Reflection: Teaching Methods for Clinical Settings.* Academic Medicine, 2002. **77**(12): p. 1185–1188.

20. Breitbart, J., Khanna, G., and Kassel, G., *An Exploration of CUDA and CBEA for a Gravitational Wave Data-Analysis Application (Einstein@Home).* ARXIV, 2009. **2**: p. 1826.

21. Bro-Nielsen, M., Inc, H., and Rockville, M., *Finite Element Modeling in Surgery Simulation.* Proceedings of the IEEE, 1998. **86**(3): p. 490-503.

22. Buck, I., Foley, T., Horn, D., et al., *Brook for GPUs: Stream Computing on Graphics Hardware.* ACM Transactions on Graphics, 2004. **23**(3): p. 777-786.

23. Buttari, A., Luszczek, P., Kurzak, J., et al. *A Rough Guide to Scientific Computing on the PlayStation 3*. 2007 [cited 2009 1-March]; Available from: http://www.netlib.org/utk/people/JackDongarra/PAPERS/scop3.pdf.

24. Cavusoglu, M.C., Goktekin, T.G., Tendick, F., et al., *GiPSi: An Open Source/Open Architecture Software Development Framework for Surgical Simulation*, in *Medicine Meets Virtual Reality 12*. 2004: Newport Beach, CA. p. 46–48.

25. Chan, S.L. and Purisima, E.O., *A New Tetrahedral Tesselation Scheme for Isosurface Generation.* Computers & Graphics, 1998. **22**(1): p. 83-90.

26. Chang, K., *Randomized Controlled Trial of Coblation versus Electrocautery Tonsillectomy.* Otolaryngology-Head and Neck Surgery, 2005. **132**(2): p. 273-280.

27. Chen, T., Raghavan, R., Dale, J., et al., *Cell Broadband Engine Architecture and Its First Implementation: A Performance View.* IBM Journal of Research and Development, 2007. **51**(5): p. 559-572.

28. Conti, F., Barbagli, F., Morris, D., et al., *CHAI 3D: An Open-Source Library for the Rapid Development of Haptic Scenes*, in *IEEE World Haptics*. 2005: Pisa, Italy.

29. Cooke, M., Irby, D., Sullivan, W., et al., *American Medical Education 100 Years after the Flexner Report.* New England Journal of Medicine, 2006. **355**: p. 1339-44.

30. Cooper, L. and Maddock, S. *Preventing Collapse Within Mass-Spring-Damper Models of Deformable Objects*. in *Fifth International Conference in Central Europe in Computer Graphics and Visualisation*. 1997. Pilsen, Czech Republic.

31. Cotin, S., Neumann, P., Wu, X., et al., *Collaborative Development of an Open Framework for Medical Simulation*, in *MICCAI Open-Source Workshop*. 2005: Copenhagen.

32. Cotin, S. and Passenger, J. *Efficient nonlinear FEM for soft tissue modelling and its GPU implementation within the open source framework SOFA.* 2008: Springer-Verlag New York Inc.

33. Crassin, C., Neyret, F., Lefebvre, S., et al., *GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering*, in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. 2009, ACM: Boston, Massachusetts. p. 15-22.

34. Crytek. *Crytek GmbH: Crytek Celebrates its 10th Anniversary*. 2009 [cited 2010 15 May]; Available from: http://www.crytek.com/news/news/?tx_ttnews[tt_news]=166&tx_ttnews[backPid]=1&cHash=d6b704eabd.

35. Davis, D.A., Mazmanian, P.E., Fordis, M., et al., *Accuracy of Physician Self-assessment Compared With Observed Measures of Competence: A Systematic Review.* JAMA, 2006. **296**(9): p. 1094-1102.

36. Desbrun, M. and Gascuel, M.P., *Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies*, in *6th Eurographics Workshop on Computer Animation and Simulation*. 1996: Grenoble, France. p. 61–76.

37. Divi, V. and Benninger, M., *Postoperative Tonsillectomy Bleed: Coblation versus Noncoblation.* The Laryngoscope, 2005. **115**(1): p. 31.

38. Dürst, M.J., *Additional Reference to Marching Cubes.* Computer Graphics, 1988. **22**: p. 72-73.

39. Edmond Jr, C., *Impact of the Endoscopic Sinus Surgical Simulator on Operating Room Performance.* Laryngoscope, 2002. **112**(7): p. 1148-58.

40. Ericson, C., *Real-time Collision Detection*. 2005: Morgan Kaufmann.

41. Ericsson, K., *Deliberate Practice and the Acquisition and Maintenance of Expert Performance in Medicine and Related Domains.* Academic Medicine, 2004. **79**(10): p. 70.

42. Erleben, K., Sporring, J., Henriksen, K., et al., *Physics-Based Animation*. 2005: Charles River Media.

43. Faletti, G. and Vezzadini, L., *NeuroVR: An Open Source Virtual Reality Platform for Clinical Psychology and Behavioral Neurosciences,* in *Medicine Meets Virtual Reality 15*, J.D. Westwood, Editor. 2007, IOS Press: Newport Beach, CA. p. 394.

44. Fanning, R.M. and Gaba, D.M., *The Role of Debriefing in Simulation-Based Learning.* Simulation in Healthcare, 2007. **2**(2): p. 115-125.

45. Faure, F., Allard, J., Cotin, S., et al., *SOFA: A Modular Yet Efficient Simulation Framework.* 2007.

46. Felder, R. and Silverman, L., *Learning and Teaching Styles in Engineering Education.* Engineering Education, 1988. **78**(7): p. 674-681.

47. Feldmann, H., *200 Year History of Tonsillectomy.* Laryngo-rhino-otologie, 1997. **76**(12): p. 751.

48. Ferguson, E., James, D., and Madeley, L., *Factors Associated with Success in Medical School: Systematic Review of the Literature.* British Medical Journal, 2002. **324**(7343): p. 952.

49. Fernando, R., *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. 2004: Pearson Higher Education.

50. Freidlin, B., *DirectX 8.0-Enhanced Real-Time Character Animation with Matrix Palette Skinning and Vertex Shaders*, in *MSDN Magazine*. 2001. p. 100-112.

51. Frisken-Gibson, S., Fyock, C., Grimson, E., et al. *Simulating Arthroscopic Knee Surgery using Volumetric Object Representations, Real-Time Volume Rendering and Haptic Feedback*. in *CVRMed-MRCAS*. 1997: Springer Verlag.

52. Gaba, D., *The Future Vision of Simulation in Health Care.* British Medical Journal, 2004. **13**(1): p. 2.

53. Gallagher, A. and Cates, C., *Virtual Reality Training for the Operating Room and Cardiac Catheterisation Laboratory.* The Lancet, 2004. **364**(9444): p. 1538-40.

54. Garland, M., Le Grand, S., Nickolls, J., et al., *Parallel Computing Experiences with CUDA.* IEEE Micro, 2008. **28**(4): p. 13-27.

55. Gilbert, E., Johnson, D., and Keerthi, S., *A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space.* IEEE Journal on Robotics and Automation, 1988. **4**(2): p. 193-203.

56. Gottschalk, S., Lin, M., and Manocha, D. *OBBTree: A Hierarchical Structure for Rapid Interference Detection*. in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 1996. San Diego, CA: ACM New York.

57. Grantcharov, T., Kristiansen, V., Bendix, J., et al., *Randomized Clinical Trial of Virtual Reality Simulation for Laparoscopic Skills Training.* British Journal of Surgery, 2004. **91**(2): p. 146-150.

58. Gregory, A., Lin, M., Gottschalk, S., et al. *A Framework for Fast and Accurate Collision Detection for Haptic Interaction*. in *SIGGRAPH*. 2005. Los Angeles, CA: ACM.

59. Gschwind, M., Hofstee, H., Flachs, B., et al., *Synergistic Processing in Cell's Multicore Architecture.* IEEE Micro, 2006: p. 10-24.

60. Haller, G., Myles, P.S., Taffe, P., et al., *Rate of Undesirable Events at Beginning of Academic Year: Retrospective Cohort Study.* British Medical Journal, 2009. **339**(1): p. 3974.

61. Harders, M., Bajka, M., Spaelter, U., et al., *Highly-Realistic, Immersive Training Environment for Hysteroscopy.* Studies in Health Technology and Informatics, 2005. **119**: p. 176.

62. Harders, M., Steinemann, D., Gross, M., et al., *A Hybrid Cutting Approach for Hysteroscopy Simulation.* Lecture Notes in Computer Science, 2005. **3750**: p. 567.

63. Hayward, V., Astley, O., Cruz-Hernandez, M., et al., *Haptic Interfaces and Devices.* Sensor Review, 2004. **24**(1): p. 16-29.

64. Hellier, D., Samur, E., and Passenger, J., *A Modular Simulation Framework for Colonoscopy using a New Haptic Device.* Studies in Health Technology and Informatics, 2008. **132**: p. 165.

65. Hikichi, T., Yoshida, A., Igarashi, S., et al., *Vitreous Surgery Simulator.* Archives of Ophthalmology, 2000. **118**(12): p. 1679.

66. Hofstee, H., *Introduction to the Cell Broadband Engine*. 2005.

67. Hudson, T., Lin, M., Cohen, J., et al. *V-COLLIDE: Accelerated Collision Detection for VRML*. in *Proceedings of the Second Symposium on VRML*. 1997. Monterey, CA: ACM.

68. Inductiveload, U. *Delaunay Triangulation.* 2010 [cited 2010 19-Jan-2010]; Available from: http://en.wikipedia.org/wiki/Delaunay_triangulation.

69. Intel, C. *Intel Xeon Processor - Intel Microprocessor Export Compliance Metrics*. 2009 08-Dec-2009 [cited 2009 6-Dec]; Available from: http://www.intel.com/support/processors/xeon/sb/CS-020863.htm.

70. James, D.L. and Pai, D.K. *ArtDefo: Accurate Real Time Deformable Objects*. in *The 26th Annual Conference on Computer Graphics and Interactive Techniques*. 1999. Los Angeles, CA: ACM.

71. James, D.L. and Pai, D.K., *Multiresolution Green's Function Methods for Interactive Simulation of Large-Scale Elastostatic Objects.* ACM Transactions on Graphics, 2003. **22**(1): p. 47-82.

72. Junker, G., *Pro OGRE 3D Programming*. 2006: Apress. 288.

73. Kennedy, D., *Functional Endoscopic Sinus Surgery: Technique.* Archives of Otolaryngology- Head and Neck Surgery, 1985. **111**(10): p. 643.

74. Khalifa, Y., Bogorad, D., Gibson, V., et al., *Virtual Reality in Ophthalmology Training.* Survey of Ophthalmology, 2006. **51**(3): p. 259-273.

75. Klein, J., Bartz, D., Friman, O., et al. *Advanced Algorithms in Medical Computer Graphics*. in *EG-STAR08 29th annual conference of the European Association for Computer Graphics*. 2008. Crete Greece.

76. Labelle, F. and Shewchuk, J., *Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles*, in *SIGGRAPH*. 2007, ACM: San Diego, CA. p. 57.

77. Larson, S., Snow, C., Shirts, M., et al., *Folding@ Home and Genome@ Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology*. ARXIV, 2009.

78. Laurell, C., Söderberg, P., Nordh, L., et al., *Computer-Simulated Phacoemulsification.* Ophthalmology, 2004. **111**(4): p. 693-698.

79. Lee, W., *The Acquisition of Clinical Ward Skills during Undergraduate Medical Training.* Journal of Medical Education, 1980. **55**(12): p. 1029-31.

80. Linked-In. *Epic Games Company Profile | LinkedIn*. 2010 [cited 2010 15-May]; Available from: http://www.linkedin.com/companies/epic-games.

81. Lohmueller, F.A. *Tetra-Lattice*. 2000 [cited 2010 14-Feb]; Available from: http://www.treeincarnation.com/images/tetra-lattice.gif.

82. Lorensen, W.E. and Cline, H.E. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. in *The 14th Annual Conference on Computer Graphics and Interactive Techniques*. 1987: ACM.

83. MacDonald, J., Williams, R.G., and Rogers, D.A., *Self-Assessment in Simulation-Based Surgical Skills Training.* The American Journal of Surgery, 2003. **185**(4): p. 319-322.

84. Marescaux, J., Clement, J., Tassetti, V., et al., *Virtual Reality Applied to Hepatic Surgery Simulation: The Next Revolution.* Annals of Surgery, 1998. **228**(5): p. 627.

85. Marks, S., Windsor, J., and Wünsche, B. *Evaluation of Game Engines for Simulated Clinical Training*. in *New Zealand Computer Science Research Student Conference*. 2008. Christchurch, New Zealand.

86. Masutani, Y., Inoue, Y., Ishii, K., et al. *Development of Surgical Simulator Based on FEM and Deformable Volume-Rendering*. in *Medical Imaging 2004*. 2004. San Diego, CA: The International Society for Optical Engineering.

87. Mayrose, J., Kesavadas, T., Chugh, K., et al., *Utilization of Virtual Reality for Endotracheal Intubation Training.* Resuscitation, 2003. **59**(1): p. 133-138.

88. McCarthy, A., Moody, L., Waterworth, A., et al., *Passive Haptics in a Knee Arthroscopy Simulator: Is it Valid for Core Skills Training?* Clinical Orthopaedics and Related Research, 2006. **442**: p. 13.

89. McCool, M. and Inc, R. *Data-Parallel Programming on the Cell BE and the GPU Using the RapidMind Development Platform*. in *The GSPx Multicore Applications Conference*. 2006. Santa Clara, CA.

90. Miller, K., Joldes, G., Lance, D., et al., *Total Lagrangian Explicit Dynamics Finite Element Algorithm for Computing Soft Tissue Deformation.* Communications in Numerical Methods in Engineering, 2007. **23**(2): p. 121–134.

91. Minor, B., Nutter, M., and Madruga, J. (2007) *IRT: An Interactive Ray Tracer for the CELL Processor*.

92. Minor, B. *RT Ray Tracing/CPU/GPU Performance - nForcersHQ.com*. 2009 [cited 2009 5-Dec-2009]; Available from: http://www.nforcershq.com/forum/rt-ray-tracing-cpu-gpu-performance-t63576.html.

93. Mitic, S., Tvinnereim, M., Lie, E., et al., *A Pilot Randomized Controlled Trial of Coblation Tonsillectomy versus Dissection Tonsillectomy with Bipolar Diathermy Haemostasis.* Clinical Otolaryngology, 2007. **32**(4): p. 261-267.

94. Molino, N., Bridson, R., and Fedkiw, R., *Tetrahedral Mesh Generation for Deformable Bodies*, in *SIGGRAPH*. 2003, SCA: San Diego, CA.

95. Monserrat, C., Meier, U., Alcaniz, M., et al., *A New Approach for the Real-Time Simulation of Tissue Deformations in Surgery Simulation.* Computer Methods and Programs in Biomedicine, 2001. **64**(2): p. 77-85.

96. Morgan and Cleave-Hogg, *Evaluation of Medical Students' Performance using the Anaesthesia Simulator*. Medical Education, 2000. **34**(1): p. 42-45.

97. Morris, D., Sewell, C., Blevins, N., et al., *A Collaborative Virtual Environment for the Simulation of Temporal Bone Surgery*, in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2004, Springer: Rennes, France. p. 319-327.

98. Morris, D., Girod, S., Barbagli, F., et al., *An Interactive Simulation Environment for Craniofacial Surgical Procedures.* Studies in Health Technology and Informatics, 2005. **111**: p. 334-341.

99. Morris, D., *Automatic Preparation, Calibration, and Simulation of Deformable Objects.* 2006.

100. Morris, D., Sewell, C., Barbagli, F., et al., *Visuohaptic Simulation of Bone Surgery for Training and Evaluation.* IEEE Computer Graphics and Applications, 2006. **26**(6): p. 48-57.

101. Morris, D. and Salisbury, K., *Automatic Preparation, Calibration, and Simulation of Deformable Objects.* Computer Methods in Biomechanics and Biomedical Engineering, 2008. **Taylor & Francis**(11): p. 3.

102. Müller, M. and Gross, M. *Interactive Virtual Materials*. in *Graphics Interface 2004*. 2004. Ontario, Canada: Canadian Human-Computer Communications Society, University of Waterloo, Canada.

103. Munshi, A. (2009) *The OpenCL Specification Version 1.0*. Khronos OpenCL Working Group.

104. Nave, R. *Elastic and Inelastic Collisions*. 2010 [cited 2010 1 March]; Available from: http://hyperphysics.phy-astr.gsu.edu/Hbase/elacol.html.

105. Nealen, A., Müller, M., Keiser, R., et al., *Physically Based Deformable Models in Computer Graphics.* Computer Graphics Forum, 2006. **25**(4): p. 809-836.

106. Nikitin, I., Nikitina, L., Frolov, P., et al. *Real-Time Simulation of Elastic Objects in Virtual Environments using Finite Element Method and Precomputed Green's Functions*. in *The Wworkshop on Virtual Environments*. 2002. Barcelona, Spain: Eurographics Association.

107. Noon, A. and Hargreaves, S., *Increased Post-Operative Haemorrhage Seen in Adult Coblation Tonsillectomy.* The Journal of Laryngology and Otology, 2006. **117**(09): p. 704-706.

108. NVidia, *Compute Unified Device Architecture Programming Guide 2.2*. 2008, NVIDIA: Santa Clara, CA.

109. NVidia, *CUDA C Programming Best Practices Guide*. 2009, NVIDIA: Santa Clara, CA.

110. Otaduy, M., Jain, N., Sud, A., et al. *Haptic Display of Interaction Between Textured Models*. in *IEEE Visualization*. 2005. Minneapolis, MN: ACM.

111. Owens, J., Luebke, D., Govindaraju, N., et al. *A Survey of General-Purpose Computation on Graphics Hardware*. in *Eurographics*. 2007. Prague, Czech Republic: Wiley.

112. Owings, M. and Kozak, L., *Ambulatory and Inpatient Procedures in the United States, 1996.* Vital and Health Statistics, 1998(139): p. 1.

113. Paradise, J., Bluestone, C., Bachman, R., et al., *Efficacy of Tonsillectomy for Recurrent Throat Infection in Severely Affected Children. Results of Parallel Randomized and Nonrandomized Clinical Trials.* New England Journal of Medicine, 1984. **310**(11): p. 674.

114. Paradise, J., Bluestone, C., Colborn, D., et al., *Tonsillectomy and Adenotonsillectomy for Recurrent Throat Infection in Moderately Affected Children.* Pediatrics, 2002. **110**(1): p. 7.

115. Parikh, S., Chan, S., Agrawal, S., et al., *Integration of Patient-Specific Paranasal Sinus Computed Tomographic Data into a Virtual Surgical Environment.* American Journal of Rhinology & Allergy, 2009. **23**(4): p. 442-447.

116. Pasquero, J. and Hayward, V. *STRESS: A Practical Tactile Display System with One Millimeter Spatial Resolution and 700 Hz Refresh Rate*. in *Eurohaptics*. 2003. Dublin, Ireland.

117. Polites, N., Joniau, S., Wabnitz, D., et al., *Postoperative Pain Following Coblation Tonsillectomy: Randomized Clinical Trial.* ANZ Journal of Surgery, 2006. **76**(4): p. 226.

118. Provot, X. *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour*. in *Graphics Interface*. 1995.

119. Riegle, E., Gunter, J., Lusk, R., et al., *Comparison of Vasoconstrictors for Functional Endoscopic Sinus Surgery in Children.* The Laryngoscope, 2009. **102**(7): p. 820-823.

120. Roberts, K., Bell, R., and Duffy, A., *Evolution of Surgical Skills Training.* World Journal of Gastroenterology, 2006. **12**(20): p. 3219.

121. Rodrigues, M., Gillies, D., and Charters, P., *Modelling and Simulation of the Tongue During Laryngoscopy.* Computer Networks and ISDN Systems, 1998. **30**(20-21): p. 2037-2045.

122. Rodrigues, M., Gillies, D., and Charters, P., *A Biomechanical Model of the Upper Airways for Simulating Laryngoscopy.* Computer Methods in Biomechanics and Biomedical Engineering, 2000. **4**(2): p. 127-148.

123. Rodrigues, M., Gillies, D., and Charters, P. *Realistic Deformable Models for Simulating the Tongue during Laryngoscopy*. in *International Workshop on Medical Imaging and Augmented Reality*. 2001.

124. Ruckley, M.R. *Functional endoscopic sinus surgery (FESS) : Patient's Guide*. 31/03/2010]; Available from: http://www.privatehealth.co.uk/private-operations/ear-nose-and-throat/functional-endoscopic-sinus-surgery-fess-/.

125. Ruffaldi, E., Morris, D., Edmunds, T., et al., *Standardized Evaluation of Haptic Rendering Systems*, in *Haptic Interfaces for Virtual Environment and Teleoperator Systems*. 2006.

126. Ryoo, S., Rodrigues, C., Baghsorkhi, S., et al. *Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA.* in *The 13th ACM Symposium on Principles and Practice of Parallel Programming*. 2008. Salt Lake City, UT: ACM.

127. Salisbury, K., Conti, F., and Barbagli, F., *Haptic Rendering: Introductory Concepts.* IEEE Computer Graphics and Applications, 2004: p. 24-32.

128. Samur, E., Flaction, L., Spaelter, U., et al. *A Haptic Interface with Motor/Brake System for Colonoscopy Simulation*. in *The 2008 Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. 2008: IEEE Computer Society.

129. Schoner, J.L., *Interactive Haptics and Display for Viscoelastic Solids*, in *Computer Graphics Group*. 2003, Universität des Saarlandes: Saarbrucken, Germany. p. 92.

130. Schwid, H.A., Rooke, G.A., Michalowski, P., et al., *Screen-Based Anesthesia Simulation With Debriefing Improves Performance in a Mannequin-Based Anesthesia Simulator.* Teaching and Learning in Medicine, 2001. **13**(2): p. 92 - 96.

131. Sedef, M., Samur, E., and Basdogan, C., *Real-Time Finite-Element Simulation of Linear Viscoelastic Tissue Behavior Based on Experimental Data.* IEEE Computer Graphics and Applications, 2006. **26**(6): p. 58-68.

132. Serra, L., Kockro, R., Guan, C., et al., *Multimodal Volume-Based Tumor Neurosurgery Planning in the Virtual Workbench*, in *Medical Image Computing and Computer-Assisted Intervention*. 1998, Springer. p. 1007-1015.

133. Seymour, N., Gallagher, A., Roman, S., et al., *Virtual Reality Training Improves Operating Room Performance.* Annals of Surgery, 2002. **236**(4): p. 458-464.

134. Seymour, N., *VR to OR: A Review of the Evidence that Virtual Reality Simulation Improves Operating Room Performance.* World Journal of Surgery, 2008. **32**(2): p. 182-188.

135. Sharp, T. *Implementing Decision Trees and Forests on a GPU*. in *The 10th European Conference on Computer Vision*. 2008. Marseille, France: Springer.

136. Shen, J. and Lipasti, M., *Modern Processor Design: Fundamentals of Superscalar Processors*. 2003: McGraw-Hill.

137. Sinclair, M.J., Peifer, J.W., Haleblian, R., et al., *Computer-Simulated Eye Surgery. A Novel Teaching Method for Residents and Practitioners.* Ophthalmology, 1995. **102**: p. 517-521.

138. Singer, M. *Bending, Breaking and Squishing Stuff*. in *Game Developers Conference*. 2006. San Jose.

139. SOFA project - (C) INRIA, M., 2008. *SOFA :: Home*. 2009 09/11/2009]; Available from: http://www.sofa-framework.org/home.

140. Srinivasan, M. and Basdogan, C., *Haptics in Virtual Environments: Taxonomy, Research Status, and Challenges.* Computers & Graphics, 1997. **21**(4): p. 393-404.

141. Stammberger, H., *Endoscopic Endonasal Surgery - Concepts in Treatment of Recurring Rhinosinusitis. Part I. Anatomic and Pathophysiologic Considerations.* Otolaryngology, 1986. **94**(2): p. 143.

142. Stone, J., Phillips, J., Freddolino, P., et al., *Accelerating Molecular Modeling Applications with Graphics Processors.* Journal of Computational Chemistry, 2007. **28**(16): p. 2618.

143. Temple, R. and Timms, M., *Paediatric Coblation Tonsillectomy.* International Journal of Pediatric Otorhinolaryngology, 2001. **61**(3): p. 195-198.

144. Terdiman, P. *Memory-Optimized Bounding-Volume Hierarchies*. 2001 May 2009; 2005-05]. Available from: http://www.codercorner.com/Opcode.pdf.

145. Terdiman, P. *RAPID Hack*. 2002 [cited 2009 13-Nov-2009]; Available from: http://www.codercorner.com/RAPID_Hack.htm.

146. Teschner, M., Heidelberger, B., Muller, M., et al., *A Versatile and Robust Model for Geometrically Complex Deformable Solids.* Computer Graphics International, 2004. Proceedings, 2004: p. 312-319.

147. Teschner, M., Kimmerle, S., Heidelberger, B., et al., *Collision Detection for Deformable Objects.* Computer Graphics Forum, 2005. **24**(1): p. 61-81.

148. Thomas, G., Johnson, L., Dow, S., et al., *The Design and Testing of a Force Feedback Dental Simulator.* Computer Methods and Programs in Biomedicine, 2001. **64**(1): p. 53-64.

149. Thomaszewski, B., Wacker, M., and Straßer, W. *A Consistent Bending Model for Cloth Simulation with Corotational Subdivision Finite Elements*. in *SIGGRAPH*. 2006. Vienna, Austria: Eurographics Association.

150. Timms, M. and Temple, R., *Coblation tonsillectomy: a double blind randomized controlled study.* The Journal of Laryngology and Otology, 2006. **116**(06): p. 450-452.

151. Timms, M.S. and Temple, R.H., *Coblation Tonsillectomy: A Double Blind Randomized Controlled Study.* The Journal of Laryngology & Otology, 2002. **116**(06): p. 450-452.

152. Tolsdorff, B., Petersik, A., Pflesser, B., et al., *Preoperative Simulation of Bone Drilling in Temporal Bone Surgery.* International Journal of Computer Assisted Radiology and Surgery, 2007. **2**: p. 160-180.

153. Van den Bergen, G., *Efficient Collision Detection of Complex Deformable Models using AABB Trees.* Journal of Graphics Tools, 1998. **2**(4): p. 1-13.

154. Van den Bergen, G., *A Fast and Robust GJK Implementation for Collision Detection of Convex Objects.* Journal of Graphics Tools, 1999. **4**(2): p. 7-25.

155. Van den Bergen, G. *Proximity Queries and Penetration Depth Computation on 3D Game Objects*. in *Game Developers Conference*. 2001.

156. Van den Bergen, G. *SOLID - Software Library for Interference Detection*. 2004 [cited 2008 17-July]; Available from: http://www.win.tue.nl/~gino/solid/index.html.

157. Vozenilek, J., Huff, J., and Reznek, M., *See One, Do One, Teach One: Advanced Technology in Medical Education.* Academic Emergency Medicine, 2004. **11**(11): p. 1149-1154.

158. Wagner, C., Lederman, S., and Howe, R. *A Tactile Shape Display using RC Servomotors*. in *The 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. 2002: IEEE Computer Society.

159. Weisstein, E.W. *Green's Function*. 2010 [cited 2010 5-Jan-2010]; Available from: http://mathworld.wolfram.com/GreensFunction.html.

160. Weller, J., Robinson, B., Larsen, P., et al., *Simulation-Based Training to Improve Acute Care Skills in Medical Undergraduates.* Journal of the New Zealand Medical Association, 2004. **117**: p. 1204.

161. Wolf, C. *IWeaver: Towards 'Learning Style'-Based E-Learning in Computer Science Education*. in *The 5th Australasian Conference on Computing Education*. 2003. Adelaide, Australia: Australian Computer Society.

162. Wormald, P., *Endoscopic Sinus Surgery: Anatomy, Three-Dimensional Reconstruction, and Surgical Technique*. 2008: Thieme.

163. Wu, W. and Heng, P., *An Improved Scheme of an Interactive Finite Element Model for 3D Soft-Tissue Cutting and Deformation.* The Visual Computer, 2005. **21**(8): p. 707-716.

164. Younis, R. and Lazar, R., *History and Current Practice of Tonsillectomy.* The Laryngoscope, 2009. **112**(S100): p. 3-5.

165. Zhou, Y., Chen, W., and Tang, Z., *An Elaborate Ambiguity Detection Method for Constructing Isosurfaces within Tetrahedral Meshes.* Computers & Graphics, 1995. **19**(3): p. 355-364.

166. Ziegler, R., Fischer, G., Müller, W., et al., *Virtual Reality Arthroscopy Training Simulator.* Computers in Biology and Medicine, 1995. **25**(2): p. 193-203.

167. Zirkle, M., Roberson, D., Leuwer, R., et al., *Using a Virtual Reality Temporal Bone Simulator to Assess Otolaryngology Trainees.* Laryngoscope, 2007. **117**(2): p. 258-263.

168. Ziv, A., Wolpe, P., Small, S., et al., *Simulation-Based Medical Education: An Ethical Imperative.* Academic Medicine, 2003. **78**: p. 783-788.

Appendix A: Mesh Coupler C++ Code Listing

```
1     __global__ void kernelUpdate(float3* deformedVertsOut,
float3* vertsIn, unsigned int* vertsToNodes, float3* blendWeights,
float3* defoGridNodes, unsigned char* mask, unsigned int numVerts)
2   {
3     const unsigned int vertId = blockIdx.x * blockDim.x +
threadIdx.x;
4
5     if (vertId >= numVerts)
6       return;
7
8     unsigned int nodeId = vertsToNodes[vertId];
9
10     if (nodeId != NULL_NODE_ID)
11     {
12       float3 n0; // origin
13       float3 w = blendWeights[vertId];
14       unsigned int adjNodeIds[6];
15       CalcAdjNodeIds(nodeId, adjNodeIds);
16
17       if (mask[nodeId]) // if not masked
18       {
19         n0 = defoGridNodes[nodeId]; // simply lookup the
location
20       }
21       else
22       {
23         // Infer the position of the origin from a set of 3
24         unsigned int x0Id = (mask[adjNodeIds[0]] ? adjNodeIds[0]
: NULL_NODE_ID);
25         unsigned int x1Id = (mask[adjNodeIds[1]] ? adjNodeIds[1]
: NULL_NODE_ID);
26         unsigned int y0Id = (mask[adjNodeIds[2]] ? adjNodeIds[2]
: NULL_NODE_ID);
27         unsigned int y1Id = (mask[adjNodeIds[3]] ? adjNodeIds[3]
: NULL_NODE_ID);
28         unsigned int z0Id = (mask[adjNodeIds[4]] ? adjNodeIds[4]
: NULL_NODE_ID);
29         unsigned int z1Id = (mask[adjNodeIds[5]] ? adjNodeIds[5]
: NULL_NODE_ID);
30
31         bool cornerFound = true;
32         float flip = -1.0f;
33         if (!mask[nodeId]) // if the origin is masked
34         {
35           // Find a triple to use to infer the position of the
origin
36           unsigned int p0Id, p1Id, p2Id;
37           if (x0Id != NULL_NODE_ID)
```

```
38          {
39            p0Id = x0Id;
40
41            if (y0Id != NULL_NODE_ID)
42            {
43              p1Id = y0Id;
44
45              if (z0Id != NULL_NODE_ID)
46                p2Id = z0Id;
47              else if (z1Id != NULL_NODE_ID)
48              {
49                p2Id = z1Id;
50                flip *= -1.0f;
51              }
52              else
53                cornerFound = false;
54            }
55            else if (y1Id != NULL_NODE_ID)
56            {
57              p1Id = y1Id;
58              flip *= -1.0f;
59
60              if (z0Id != NULL_NODE_ID)
61                p2Id = z0Id;
62              else if (z1Id != NULL_NODE_ID)
63              {
64                p2Id = z1Id;
65                flip *= -1.0f;
66              }
67              else
68                cornerFound = false;
69            }
70            else
71              cornerFound = false;
72          }
73          else if (x1Id != NULL_NODE_ID)
74          {
75            p0Id = x1Id;
76            flip *= -1.0f;
77
78            if (y0Id != NULL_NODE_ID)
79            {
80              p1Id = y0Id;
81
82              if (z0Id != NULL_NODE_ID)
83                p2Id = z0Id;
84              else if (z1Id != NULL_NODE_ID)
85              {
86                p2Id = z1Id;
87                flip *= -1.0f;
88              }
89              else
90                cornerFound = false;
91            }
92            else if (y1Id != NULL_NODE_ID)
93            {
94              p1Id = y1Id;
95              flip *= -1.0f;
96
97              if (z0Id != NULL_NODE_ID)
```

```
98                p2Id = z0Id;
99              else if (z1Id != NULL_NODE_ID)
100             {
101               p2Id = z1Id;
102               flip *= -1.0f;
103             }
104             else
105               cornerFound = false;
106           }
107           else
108             cornerFound = false;
109         }
110         else
111           cornerFound = false;
112
113         if (cornerFound)
114         {
115           float3 p0 = defoGridNodes[p0Id];
116           float3 p1 = defoGridNodes[p1Id];
117           float3 p2 = defoGridNodes[p2Id];
118
119           float3 e1 = p1 - p0;
120           float3 e2 = p2 - p0;
121           float3 n = cross(normalize(e1), normalize(e2));
122           float3 c = (p0 + p1 + p2) / 3.0f;
123           n0 = c + flip * n;
124         }
125         else
126         {
127           cornerFound = true;
128
129           if (x0Id != NULL_NODE_ID && x1Id != NULL_NODE_ID)
130             n0 = 0.5f * (defoGridNodes[x0Id] +
defoGridNodes[x1Id]);
131           else if (y0Id != NULL_NODE_ID && y1Id !=
NULL_NODE_ID)
132             n0 = 0.5f * (defoGridNodes[y0Id] +
defoGridNodes[y1Id]);
133           else if (z0Id != NULL_NODE_ID && z1Id !=
NULL_NODE_ID)
134             n0 = 0.5f * (defoGridNodes[z0Id] +
defoGridNodes[z1Id]);
135           else
136             cornerFound = false;
137         }
138
139         if (!cornerFound)
140         {
141           if (   x0Id != NULL_NODE_ID)
142             n0 = defoGridNodes[x0Id] - make_float3(1.0f,
0.0f, 0.0f);
143           else if (x1Id != NULL_NODE_ID)
144             n0 = defoGridNodes[x1Id] + make_float3(1.0f,
0.0f, 0.0f);
145           else if (y0Id != NULL_NODE_ID)
146             n0 = defoGridNodes[y0Id] - make_float3(0.0f,
1.0f, 0.0f);
147           else if (y1Id != NULL_NODE_ID)
148             n0 = defoGridNodes[y1Id] + make_float3(0.0f,
1.0f, 0.0f);
```

```
149             else if (z0Id != NULL_NODE_ID)
150               n0 = defoGridNodes[z0Id] - make_float3(0.0f,
0.0f, 1.0f);
151             else if (z1Id != NULL_NODE_ID)
152               n0 = defoGridNodes[z1Id] + make_float3(0.0f,
0.0f, 1.0f);
153           }
154         }
155       }
156
157       // If the adjacent node is present, use it to compute the
x-axis
158       float3 xAxis = make_float3(1.0f, 0.0f, 0.0f);
159       if (adjNodeIds[0] != NULL_NODE_ID && mask[adjNodeIds[0]])
160         xAxis = defoGridNodes[adjNodeIds[0]] - n0;
161       else if (adjNodeIds[1] != NULL_NODE_ID &&
mask[adjNodeIds[1]])
162         xAxis = n0 - defoGridNodes[adjNodeIds[1]];
163       //else
164       // return;
165
166       float3 yAxis = make_float3(0.0f, 1.0f, 0.0f);
167       if (adjNodeIds[2] != NULL_NODE_ID && mask[adjNodeIds[2]])
168         yAxis = defoGridNodes[adjNodeIds[2]] - n0;
169       else if (adjNodeIds[3] != NULL_NODE_ID &&
mask[adjNodeIds[3]])
170         yAxis = n0 - defoGridNodes[adjNodeIds[3]];
171       //else
172       // return;
173
174       float3 zAxis = make_float3(0.0f, 0.0f, 1.0f);
175       if (adjNodeIds[4] != NULL_NODE_ID && mask[adjNodeIds[4]])
176         zAxis = defoGridNodes[adjNodeIds[4]] - n0;
177       else if (adjNodeIds[5] != NULL_NODE_ID &&
mask[adjNodeIds[5]])
178         zAxis = n0 - defoGridNodes[adjNodeIds[5]];
179       //else
180       // return;
181
182       // 2x vertId needed since each vertex has position and
normal
183       // TODO: Use 3D span and 3D min
184       deformedVertsOut[2 * vertId + 0] = ((n0 +
(float)dMechConfig.dimX * (w.x * xAxis + w.y * yAxis + w.z *
zAxis)) * dCouplerConfig.vertsSpanX + dCouplerConfig.vertsMinX) /
(float)dMechConfig.dimX;
185       float3 norm = vertsIn[2 * vertId + 1];
186       // Warp the normal according to the deformation
187       deformedVertsOut[2 * vertId + 1] = norm.x * xAxis +
norm.y * yAxis + norm.z * zAxis;
188     }
189   }
```

Appendix B: An Earlier Version of ISim

Prior to developing the simulator described in 9.2 another endotracheal intubation simulator was developed. This earlier version uses novel image-based collision and deformation effects (similar to that described by Otuday *et al*. [110]) to simulate the key interaction with the simulated patient's tongue. Features of the simulation include the ability of the virtual patient to exhibit the discolouration (bluing) of the skin and lips due to hypoxia (Figure 89), and simulation of damage to the upper teeth that can be chipped if the laryngoscope collides with them or is pressed slowly against them with sufficient force. This allows students to observe the effects of this common mistake without the real-world consequences.

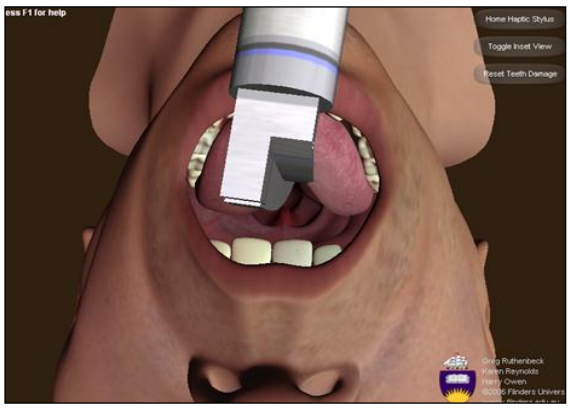


Figure 88: An earlier version of ISim

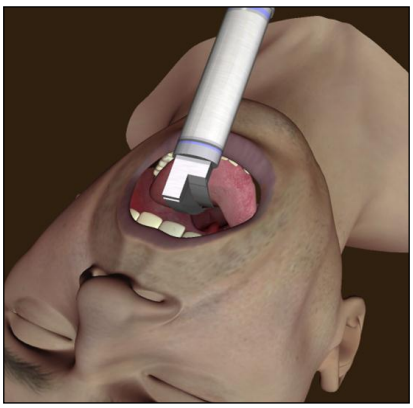


Figure 89: Bluing of lips and skin indicate lack of oxygen in the blood

Image-based Collision Detection and Deformation

Real time physically based models are acknowledged as computationally expensive and are generally relatively complex. So, rather than model the mechanics of the tongue, I have developed a method that requires substantially less

processing to provide a simple tongue deformation effect that can also be haptically rendered.

The effect uses low-resolution depth images that can be created even with older GPUs (since no special shader stages or instructions are used). Processing load is invariant and mainly dependent on the resolution of the images used. The only requirements is that the GPU provides support for render-to-texture of per-pixel depth information (either directly from the z-buffer or using a specialised shader program). Additionally, in order to support haptic rendering, the render-target texture format must also be CPU lockable (such that it can be read by the CPU) so that reactive forces can be computed and delivered to the haptic device.

Effect Overview

1. Update cameras
2. Render depth (z-values) to textures
3. Compute intersection volume
4. Apply deformation to tongue model
5. Haptic Render

It is critical that the virtual viewpoints used to capture the depth images are correctly positioned. Two orthogonal cameras were used such that they both look directly back at each other. One is aimed at the tongue from the direction of the laryngoscope, the other is aimed at the laryngoscope blade from the tongue. The cameras are defined such that the near and far planes of the view frustums are perpendicular to the line connecting the centre of the laryngoscope blade to the centre of the tongue. Figure 90 shows a teapot model intersecting the tongue model. The depth information from each of the cameras is shown; the tongue depth is shown in red, and the teapot depth in green. Simply subtracting the values of the two colour channels of this image defines the intersection volume.

Applying the computed intersection volume directly to the tongue vertices results in a hard edged gouge, whereas the simulation requires a smooth deformation effect. To correct this, the intersection volume texture is down-sampled and blurred. This is trivial using the GPU because it uses very commonly used operations that are highly optimised. The softened edges result in a smooth transition from maximum offset to un-displaced vertices (see Figure 91).

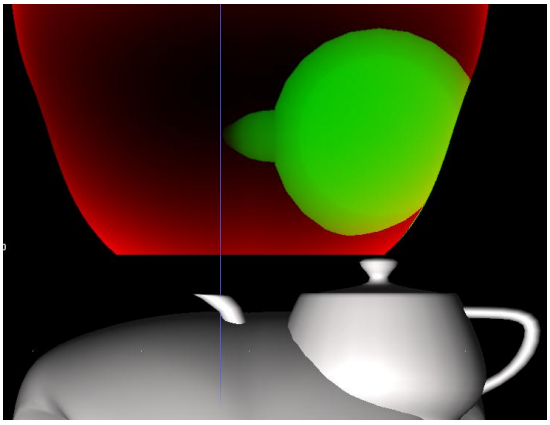


Figure 90: Intersection volume from depth images

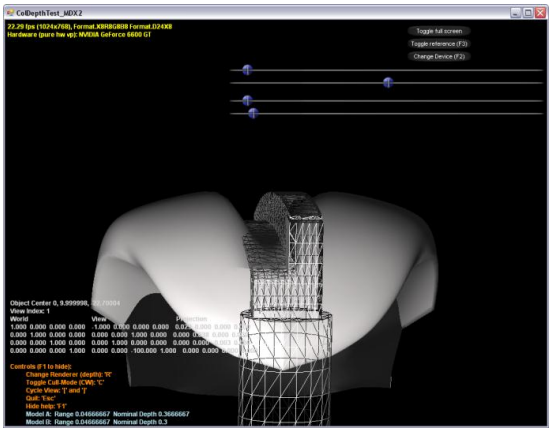


Figure 91: Laryngoscope deforming tongue model

Combining the depth textures results in the intersection volume. Mapping the texture coordinates back to the tongue model enables the vertices of the tongue mesh to be offset using the vertex shader. Hence, the tongue deforms in response to contact with the laryngoscope, based entirely on simple image-based operations (Figure 91). Additional realism is achieved by applying the offset to vertices on the CPU, which allows adjacency information to be used to update the vertex normals and improves the accuracy of the shading. Further investigation into offsetting normals based on the intersection volume texture would be more effective but has not yet been investigated.

This approach can be extended by introducing an intersection-texture stack. The stack could be used to store the intersection volume (and consequent deformation) at regular time intervals. These textures, together with the camera locations (to define the axis of interaction), would enable visco-elastic behaviour to be modelled; the texture stack would enable prior displacement to persist after the objects are no longer touching.

Haptic Rendering

Haptic rendering can be complex. However, this technique provides a very simple method for detecting collisions, deforming interacting structures, and haptic rendering. The reactive force is computed using the assumption that the force is proportional to the intersection volume. This volume can be obtained trivially by

totalling the pixel intensities of the intersection volume texture. Other relationships between the intersection volume and the reactive force can be delivered by using the volume to lookup values of another curve. This technique was successfully used to remove haptic kicks (see section 8.6.1.4) that occur when the intersection volume is small while retaining progressive increases in reactive force when the intersection is larger.

This technique is well suited to simple convex shapes. However, adding physically based constraints such as accurate volume preservation would be difficult using this method. This version of ISim was developed using C# and DirectX 9. With further development, image-based techniques could provide a very efficient method for approximating deformation and perform simple haptic rendering.