# HAPTICS AND PHYSICAL SIMULATION

# FOR VIRTUAL BONE SURGERY

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Dan Morris

August 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
(Kenneth Salisbury) Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
(Sabine Girod)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
(Federico Barbagli)

Approved for the University Committee on Graduate Studies.

# Abstract

Surgical training has traditionally followed an apprenticeship model: residents observe a number of procedures, then begin practicing in the operating room. And every time a resident practices his first procedure on a patient, that patient is put at some level of additional risk. Even in specialties where cadaver training is applicable (a restricted set to begin with), cadavers are expensive, are available only in limited number, and lack the physiology that guides surgical decision-making. Thus the effectiveness of cadavers in preparing residents for surgical practice is limited.

Fortunately, computer-based simulation offers an intermediate between observation and live-patient operation. Virtual environments can allow residents to practice both basic skills and procedural logic at extremely low cost, allowing the presentation of a wide variety of operating-room scenarios that cannot be duplicated in cadaver labs. Furthermore, computer-based simulation can offer even experienced surgeons a chance to practice infrequently-performed procedures, to learn new surgical techniques, and to rehearse procedures preoperatively on patient-specific anatomy. An analogy can be made to the highly successful field of flight simulation, which has been routinely used to train and re-educate pilots for decades.

However, significant technical challenges stand between today's surgical simulation systems and the virtual operating room that will become a standard part of tomorrow's medical training. Simulators are still limited in rendering quality, immersiveness, intuitiveness, and simulation realism. This thesis addresses some of those challenges, specifically in the context of simulating procedures performed on the temporal bone and mandible.

We present an overview of our simulation environment, specifically focusing on how this software delivers the sources of intraoperative feedback that are relevant to training surgical skills. We then discuss a project inspired by this environment, which asks whether haptic feedback can be used to *teach* motor skills, adding a level of training not available in traditional training labs. We then address one of the most difficult problems in surgical simulation: effectively simulating realistic deformable materials. Specifically, we address the adjustment of an interactive, low-computational-cost deformation model to behave like a more complex model. We then present a series of algorithms and data structures that emerged from this work, and conclude with a discussion on the evaluation of the *realism* of haptic rendering systems.

The design and implementation of our simulator has proceeded in close collaboration with surgeons, and we have designed each component to fill a niche that was found to be relevant in building a practical surgical simulator. This dissertation demonstrates the effectiveness of this collaborative, multidisciplinary approach to the design of medical simulators.

# Acknowledgments

Thanks go out to my advisor, Ken Salisbury, for allowing me the freedom to pursue my own directions on this project throughout my time at Stanford. I also thank the rest of the BioRobotics lab, especially Chris Sewell and Nik Blevins, my primary collaborators on the simulation project. And a particularly deep thank-you to Federico Barbagli, for his constant guidance, numerous lattés, and ideas ranging from practical to way-out-there. Other close collaborators I would like to acknowledge include Hong Tan, Sabine Girod, Emanuele Ruffaldi, Tim Chang, Doug Wilson, Derek Gaw, Dinesh Pai, and Tim Edmunds.

Thanks also to CyberKinetics and the Brown University Department of Neuroscience, for giving me a home away from home (in an intellectual sense) and a concrete tie to the other end of translational research.

Funding for this work was provided by the National Defense Science and Engineering Graduate Fellowship, and later by the AO Foundation and NIH grant LM07295. I appreciate the generosity of all of these sources.

Thanks to the Robotics Lab support staff (Hoa in particular); it does not pass unnoticed that the lab would collapse without you.

Thanks of course to gLab, my home away from home (in a hanging-out sense). I'd especially like to thank the Cookie Office (a special thanks to the cookies themselves) and the gSlackers (Dave Akers, Mike Cammarano, Billy Chen, Jim Chow, Kayvon Fatahalian, Gaurav Garg, Daniel Horn, Mike Houston, Neel Joshi, Jeff Klingner, Ren Ng, Doantam Phan, Augusto Roman, Bennett Wilburn, and Ron Yeh).

I'd like to thank my family, who supported my last-minute change of career direction right *after* I finished applying to medical school (which I don't doubt now would have been a *disaster*).

Also thanks to everyone who gave me so many constructive distractions outside of work that I almost don't want to write this thesis and move away: Careless Hearts, The Sound Effect, Sachin Premasuthan, cs-ultimate, im-hoops, soccerforfun, various CS IM teams (go CS!), the California weather, etc.

Finally, a `pow("thank you",DBL_MAX)` to my girlfriend→fiancée→wife, best friend, roommate, and co-dog-sitter Merrie. Grad school is a whole lot more fun when you have someone to go through it with, and fun aside, it's fairly clear to everyone who knows us that I would *never* be graduating without you.




# Contents

# List of Tables

# List of Figures

# 1 Introduction

**Figure 1. A surgeon demonstrates drilling technique to a trainee using our networked simulation environment.**

This dissertation will present techniques for haptic rendering and physical simulation, specifically targeted toward solving problems relevant to virtual surgery. We will begin by exploring five problems faced by the surgical community and possible simulation-based solutions to those problems, to motivate the remainder of the thesis.

The first relevant challenge faced by the surgical community is the risk incurred by patients when a resident first conducts a procedure. Surgical training programs are traditionally based primarily on observation of experienced surgeons. Residents are provided with some classroom training, but the core of a resident training program is time spent observing and assisting in the OR. In certain

specialties, residents also practice on cadavers or animals, but these approaches virtually never supplant patient-centric learning due to cost and inherent dissimilarity from the procedures being trained: cadavers lack physiology, and animals are in most cases sufficiently different from humans in anatomy, physiology, and pathology to prevent fine-tuning of surgical skills. The primary drawback to this model for surgical training is the intrinsic risk at which patients are placed when a resident first practices a procedure on a live patient. Despite extensive preparation through textbooks and observation, sensorimotor skills take time to develop, and a first-time surgeon is unlikely to be as effective as an experienced superior.

A second problem facing surgical training is that most resident training programs currently lack a formal mechanism for evaluating resident progress. It is generally up to the discretion of instructors to determine when residents are prepared for various stages of intraoperative participation, a subjective system which is difficult to standardize across institutional or national boundaries.

A third challenge faced by the surgical community is the lack of a consistent mechanism for incorporating new technologies into surgical practice. Surgery is already being transformed by computer-based techniques such as robotic surgery ([62], [102], [139], [93], [120], [122], [35]) and image-guided/augmented-reality ([108], [106], [141], [57]) surgery. However, integration of new techniques and devices is still based largely on proprietary materials created by medical device manufacturers, which can be difficult to evaluate and difficult to disseminate. Furthermore, even experienced surgeons face the problem of learning to use new technologies, and the community as a whole faces the problem of rapidly developing surgical techniques, outcome metrics, and training guidelines for new devices and treatments.

A fourth challenge faced by the surgical community is the lack of a mechanism for "refreshing" surgeons – even experienced surgeons – on rarely-performed procedures, approaches, pathologies, or adverse intraoperative events. Although surgeons in the U.S. are required to participate in continuing medical education on an annual basis, this generally focuses on new techniques and does not include hands-on

review of uncommon procedures. This is especially relevant for ER and general surgeons, who see a wide variety of cases and often have little time to prepare for a case. To make the analogy to the highly-successful field of flight simulation, a pilot can fly a 747 for twenty years and never experience an engine failure, but still must be prepared to respond when one occurs.

And yet a fifth challenge faced by surgeons is the tremendous anatomic and physiological variation among patients, requiring significant adjustment and navigational decision-making intraoperatively. 3D imaging offers surgeons a preoperative view of a particular patient's anatomy, but 3D imaging is still used only sparsely in preoperative preparation (surgeons still primarily rely on 2D slices from 3D data sets). This is largely due to the inability of current 3D image viewers to replicate the approaches, perspectives, and interactions surgeons experience intraoperatively. In this sense, 3D viewers do not offer significantly more than the basic structural information surgeons currently obtain from 2D slice images. Furthermore, even an ideal image viewer would not allow surgeons to practice the difficult physical manipulations that may be required for a particular patient.

We have now seen five challenges faced by today's surgical community: risk posed by training inexperienced surgeons, evaluating resident progress, incorporating new technologies, re-training experienced surgeons, and a lack of patient-specific rehearsal techniques. Surgical simulation (a term we will use interchangeably with "virtual surgery") – particularly *haptic* surgical simulation – offers promising solutions to each of these problems.

Haptic virtual environments will allow residents to train on numerous complete procedures before ever entering an operating room, providing a strong command of the necessary sensorimotor skills and strong preparation for adverse events. Computer-based simulation can offer reliable, repeatable, automated mechanisms for evaluating resident progress, which can easily be standardized across institutions. New surgical devices and approaches will be incorporated into simulation environments before being incorporated into regular clinical use, allowing surgeons a

chance to learn and experiment, and to provide feedback to manufacturers to iteratively improve devices before high-risk clinical testing even begins. Surgeons will be able to rehearse rarely-performed procedures either regularly (as part of continuing medical education requirements) or as needed (immediately before performing an unfamiliar procedure). And finally, surgeons will be able to rehearse a procedure on a specific patient's anatomy, pathology, and even physiology in simulation before seeing that patient in the OR; this will help optimize surgical plans and minimize unexpected intraoperative events. This type of rehearsal environment is unlikely to replicate a complete (and lengthy) procedure; the most effective rehearsal will likely sit somewhere between complete simulation and interactive visualization.

And perhaps the most valuable – and most overlooked – possibilities offered by virtual surgery are those provided to personnel other than trained surgeons. For example, virtual surgery offers tremendous possibilities to veterinary surgeons, who occasionally have to operate on species they have not previously encountered, where acquiring a complete knowledge of the relevant anatomy and physiology may not be possible in the available time.

As another example, in times of crisis, it may become necessary for non-surgeons – e.g. non-surgical physicians, nurses and PA's, battlefield personnel, etc. – to perform surgery, and simulation offers a rapid and highly focused training mechanism.

Virtual surgery can also offer medical students an opportunity to explore different surgical disciplines before committing to a specialty, to evaluate their interest and the appropriateness of their skill set or to demonstrate their proficiency to residency programs.

And finally, virtual surgery – though perhaps not in its most realistic form – may be an appropriate mechanism for a *patient* to better inform himself about a procedure for which he is preparing or a procedure he has recently undergone. Most patients – even technologically informed patients – experience surgery with only a limited amount of understanding about the procedural details. While some patients

may be content with this level of information, many patients (or their families) would benefit from a deeper understanding of a procedure that may come from a video-game-like simulation environment.

Despite the numerous motivations provided above for incorporating surgical simulation into standard medical practice, and despite the commercial availability of several simulators (e.g. [182], [36], [87], [183], [191], [213]), many of which have undergone successful validation studies (e.g. [53], [215], [161], [146], [207], [12]), virtual surgery has yet to become a part of the medical mainstream.

This is due in large part to the limited realism of simulation environments, which often restricts successful simulation-based learning to basic skills training that does not depend on graphical realism or even a medical context for the virtual environment. Training for basic sensorimotor skills in laparoscopy, abstracted away from whole-procedure simulation, has thus been particularly successful [215].

In order for surgical simulation to develop into a core component of medical practice and offer the full complement of benefits outlined above, further basic research is required in the areas of graphic and haptic rendering techniques, assessment mechanisms for rendering accuracy and simulator validity, automated evaluation mechanisms, computer-assisted pedagogy, user interface design for surgical simulators, image processing for data preparation, etc.

This thesis addresses several of these issues in the context of developing a simulation environment for surgical procedures of the temporal bone (Figure 2) and mandible. The design of this environment proceeded in close collaboration with surgeons, and the sections of this thesis address individual problems or opportunities that arose during this development process.

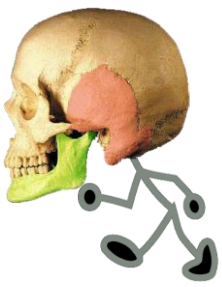


**Figure 2. Lifelike and anatomically-accurate illustration of the location of the temporal bone (red) and mandible (green).**

## 1.1 Contributions

This dissertation presents the details of our surgical simulation environment, and a series of technical advances that were made in the process of developing this environment. The key contributions of this thesis are:

**Algorithms and rendering techniques for haptic surgical simulation:** We present the haptic rendering techniques used in our surgical simulator (Section 3), which are applicable to a variety of surgical specialties and non-medical haptic applications. We also present a series of algorithms and data structures that are used in processing and preparing the data used in our simulator (Sections 5 and 6), presented in a general context that is not restricted to their application to virtual surgery. Additionally, we present mechanisms for comparing force trajectories (Section 4) and haptic rendering algorithms (Section 7) that will enable a variety of haptics and psychophysical experiments. The common threads among the presented techniques are (a) replicating and assessing the sensory feedback required for effective surgical training and (b) data-driven methods for simulation and haptic rendering.

**Experiments and experimental results:** We present two experiments involving human subjects. The first (Section 3) demonstrates the construct validity of our simulation environment and will serve as a template for future construct validity experiments. The second (Section 4) demonstrates the utility of haptic feedback in teaching force patterns, and introduces novel analysis techniques that will generalize to other learning and psychophysics experiments. We further present computational experiments evaluating algorithmic performance and accuracy in Section 5, Section 6, and Section 7.

**Software:** The work presented in this thesis has generated a body of software that will contribute to the haptics and medical simulation communities. The simulation

environment presented in Section 3 is currently in use in the Department of Head and Neck Surgery at Stanford and will continue to be a testbed for surgical simulation experiments. The software packages, code, and/or data presented in each of the other sections are available online; links are provided as each software component is discussed.

## 1.2 Dissertation Roadmap

The remainder of this dissertation is organized as follows:

In Section 2, we discuss related literature, including work on virtual surgery, haptic rendering, and physical simulation.

Section 3 describes our surgical simulation environment in detail, with a particular emphasis on haptic rendering techniques and the replication of relevant sources of intraoperative feedback.

Section 4 presents an experiment conducted to evaluate the possibility of *teaching* sequences of forces using haptic feedback. This section discusses experimental design, analysis techniques, and experimental results. Results indicate that haptic feedback can enhance learning when coupled with visual feedback.

Section 5 presents techniques for mesh generation, calibration to a finite element reference model, and interactive simulation. These techniques are presented in the context of a processing pipeline for preparing and interactively simulating deformable objects.

Section 6 presents three algorithms and data structures that contributed to the work presented in Section 3, Section 4, and Section 5. In particular, this section discusses

techniques for distance field generation, data logging for performance-sensitive multithreaded applications, and haptic curve constraints.

Section 7 presents techniques for evaluating the realism of haptic rendering algorithms; forces generates by a haptic rendering system are compared with ground truth data.

Section 8 concludes with lessons learned from our experiments in surgical simulation and discusses future work in some of the areas discussed throughout the thesis.

# 2 Related Work

This section provides an overview of related literature, particularly focusing on projects related to virtual surgery. More detailed discussions of related work are included in each subsequent section, placed specifically in the context of the work presented in that section.

## 2.1 Haptics for Virtual Surgery

Before embarking on the development of a haptic simulation environment, it is relevant to ask whether haptic feedback is relevant to surgery at all. Recent studies have begun to explore this question in physical models of surgical environments. Wagner et al [206] asked subjects to dissect a physical model of an artery with and without force feedback, and found that force feedback significantly reduced the number of errors and the overall level of applied force. Tholey et al ([200], [85]) asked subjects to perform a soft-tissue identification task in a physical model, and found that haptic feedback significantly enhanced subjects' ability to distinguish among tissue types. Kazi ([94]) found that force feedback reduces applied forces in a catheter insertion task. These results confirm the intuition that haptic feedback is critical to the fine dexterous manipulation required for surgery. The recent adoption of robotic platforms – which currently *lack* force feedback – will offer a future testing ground for the role of haptics as force-feedback capabilities are added to surgical robots.

The apparent utility of haptics in surgery suggests that effective surgical simulators will also include haptic feedback. Thus numerous surgical simulation environments have included haptics. Laparoscopic surgery has been a particularly appealing target for simulation-based learning, given the difficult learning curve for laparoscopic instruments and the reproducibility of the intraoperative field of view; environments for training laparoscopic skills constitute the bulk of simulators developed to date. Webster et al [209] present a haptic simulation environment for laparoscopic cholecystectomy, and Montgomery et al [124] present a simulation environment for laparoscopic hysteroscopy; both projects focus on haptic interaction with deformable tissue. Cotin et al [43] present a haptic simulator for interventional cardiology procedures, incorporating blood flow models and models of cardiopulmonary physiology. De et al [49] apply the method of finite spheres to a haptic simulator for laparoscopic GI surgery.

Several commercial simulators also include haptic feedback. For many of these products, initial validation studies have been performed to evaluate the efficacy of haptic simulation as a training technique. Wong et al [212] evaluated the construct validity of the Endovascular AccuTouch system (Immersion Medical) for pacemaker implantation simulation; this study differentiated participants according to their surgical experience level (similar in scope to the study presented in Section 3 of this dissertation). Engum et al [54] explored the training benefits of the CathSim simulator for intravenous catheterization, and found similar skill demonstration in participants trained using traditional methods and those using the simulator; in several methodological aspects of the task (e.g. documentation), the non-simulator group was found to be superior. Grantcharov et al [68] confirmed the construct validity (ability to differentiate users according to experience level) of the GI Mentor II system (Simbionix Ltd.), a haptic simulator for GI endoscopy. Similarly, McDougall et al [114] confirmed that construct validity of the LAPMentor system (Simbionix Ltd.), a haptic trainer for basic laparoscopic skills.

## 2.2 Related Simulation Environments

This dissertation focuses on our environment for simulating bone surgery, so we will elaborate specifically on simulation work in this area.

Several groups have developed simulators for temporal bone surgery, though none have previously demonstrated construct validity and none have been formally incorporated into surgical training programs. Bryan et al [34] present a visuohaptic environment for simulating temporal bone surgery, and the developers of this project are currently assembling a multi-institution study to validate and disseminate their work [84].

Agus, Prelstaff, Giachetti et al present a series of papers describing their simulation environment for virtual temporal bone surgery ([9], [8], [10], [7], [6]). They provide further detail on their particle model for simulating bone dust [63], their approach to haptic rendering [5], and their approach to volume rendering [11]. They present a related series of psychophysical experiments on haptic contrast sensitivity and users' abilities to differentiate bone tissue types in their simulator in [30]. Also, the same group presents an experimental approach to tuning the haptic feedback in their environment [4].

Pflesser, Petersik et al report on an environment for virtual temporal bone surgery ([152], [151], [153], [155]), focusing on haptic rendering and their adaptation of the Voxel-Point-Shell method [116] to bone surgery.

Previous work on simulating craniofacial surgery has focused largely on soft-tissue modeling for predicting post-operative facial appearance. The general paradigm is to acquire a pre-operative model of a patient's soft tissue via image segmentation or range scans, and couple that to a pre-operative model of the same patient's bone structure acquired via MR or CT. The bone model is then manipulated interactively by a surgeon, and the system attempts to use soft-tissue deformation simulation (not generally in real time) to predict facial tissue movement and post-operative appearance. Keeve et al [96] introduce this approach using laser range scans and use

the finite element method to compute deformation; a similar FEM-based approach is used by Berti et al [21]. Teschner et al ([198], [196], [197]) also use range scans as preoperative data, and use an optimization approach to model deformation via a mass-spring system.

Several previous projects have also been targeted at developing an interactive environment for manipulating bone fragments. Berti et al [21] allow the user to explicitly specify polygonal cut surfaces and provide visualization tools to assist in visualizing those cuts. Everett et al [56] provide interactive collision detection during the manipulation of bone fragments, and incorporate cephalometric labeling into their environment. Pintilie et al [157] focus on resampling, refining, and re-meshing a surface mesh to implement a cutting operation using a scalpel-like tool on the surface of a bone model.

## 2.3 Evaluating Simulation Environments

As surgical simulation environments mature and enter the medical mainstream, formal evaluation and validation will become critical aspects of simulator development, clinical approval, and marketing. Thus Section 3 of this thesis evaluates the construct validity of our simulation environment. Here we will review the recent trend in the simulation community toward evaluation studies. Several such studies – focusing on commercial simulation environments that include haptics – were also discussed above in Section 2.1.

Additionally, Youngblood et al [215] compared computer-simulation-based training to traditional "box" (mechanical simulator) training for basic laparoscopic skills, and found that trainees who trained on the computer-based simulator performed better on subsequent porcine surgery. Hariri et al [75] evaluated a simulation environment not for its ability to teach surgical skills, but for its ability to teach shoulder anatomy, and found it to be superior to textbook-based training. Srivastava et al [185] confirmed the construct validity of a simulator for arthroscopic procedures, and Van Sickle et al [204] confirmed the construct validity of the ProMIS simulator for basic laparoscopic skills. Seymour et al [180] and Grantcharov et al [69] both

evaluated the transference of laparoscopic skills for minimally-invasive cholecystectomy from a simulator to a clinical OR, and found a significant benefit from virtual training in terms of time, error rates, and economy of movement. Haluck et al [73] address the validity of the Endotower (Verefi Technologies, Inc.), a simulator for laparoscopic endoscopy.

## 2.4 Physical Simulation for Virtual Surgery

The most challenging technical problem in simulating most surgical procedures is the computation of deformation from forces. This poses the hardest case of the physical deformation problem, requiring both realism (materials must behave like the physical objects they're representing) and interactivity (deformation must be computed at rates sufficient for graphic – and in some cases haptic – rendering). This dissertation addresses this problem in Section 5, using optimization techniques to extract maximally realistic behavior from interactive simulation techniques.

A significant amount of work has been done on physical simulation of deformable materials for computer graphics, but these problems are rarely subject to interactivity constraints and can generally be manually calibrated to express a range of desired material properties. Gibson and Mirtich [64] provide a comprehensive review of the fundamental techniques in this field; this section will specifically discuss the application of physical simulation techniques to interactive deformation for virtual surgery.

Early simulation environments (e.g. [123], [33]) generally employed the network of masses and springs to model deformation. This approach is extremely fast, has extensively-studied stability properties, parallelizes extremely well, handles topology changes trivially, and is intuitive to code and extend. Unfortunately, mass-spring systems are not calibrated in terms of intuitive physical parameters, do not generally provide volume-preserving properties, are subject to instabilities when integrated explicitly, and do not generally provide physically-accurate behavior. Later work coupled traditional mass-spring systems with implicit solvers to improve stability and accuracy ([210], [209]).

More recent medical simulation environments have incorporated finite-element modeling, which provides significantly more accuracy than mass-spring systems at the expense of computational cost. Such systems generally incur a particularly high cost for topology changes; thus much of the work in this area has focused on building interactive finite element simulations that efficiently handle topology changes ([142], [105], [20], [121]).

Additionally, a number of novel deformation models have been developed specifically for medical simulation. Balaniuk and Salisbury present the Long Elements Method [15] and the Radial Elements Method [16]; both are constitutive, quasi-static methods that provide volume preservation but limit dynamic behavior. Cotin et al [44] use the finite element method as a preprocessing step and use a simpler elastic model to adjust precomputed force/response functions to interactive stimuli. More recently, the increasing availability of parallel architectures has spurred the development of simulation techniques that parallelize more naturally, including meshless techniques ([49], [50]) and parallel mass-spring systems ([133], [134]).

Although there has yet to be a consensus on a "correct" deformation model for medical simulation, effective application of any model will require accurate descriptions of the tissues represented in the simulation. Furthermore, a thorough evaluation of a deformation model requires ground truth data to which one can compare results obtained in simulation. For both of these reasons, a significant body of work has attempted to obtain material properties for physical tissues. Kerdok et al [97] collect strains throughout a deformable body, aiming to establish a standard to measure soft-tissue deformation models. Samani et al [168] measure the ex vivo response of tissue samples to applied forces. Tay et al [194] and Brouwer et al [32] perform similar measurements in vivo.

# 3 Visuohaptic Simulation of Bone Surgery

This section details the simulation environment we have developed for simulating bone surgery. We present relevant simulation techniques and describe the architecture that motivates the remainder of the thesis. A particular emphasis is placed on providing the *sensory cues* that are relevant to surgical training in these disciplines. That is, rather than striving primarily for an aesthetic sense of graphical realism, we examine the key skills that an ideal simulator would train, and the key sources of feedback that are relevant to surgical decision-making. We then provide appropriate representations of those elements in our environment. These critical aspects of the simulation environment have been identified through iterative design and prototyping in close collaboration with surgeons at Stanford.

This work has not yet been validated in a clinical setting; a clinical trial is beyond the scope of this thesis. However, the surgical simulation community defines several levels of preclinical validity, and we present an experiment here that assesses the *construct validity* of our environment. We demonstrate with statistical significance that surgeons perform better in our environment than non-surgeons (Section 3.3).

At the time of publication of this thesis, the environment described here has been installed in the resident training facility in the Department of Head and Neck Surgery at Stanford, and is being used regularly by surgeons to iteratively improve the environment. Our goal is to integrate it into the resident training program in the coming months.

Work described in this section has been published in [128], [126], and [127]. The environment presented here was also used as the infrastructure for the work presented in [179], [178], [177], and [176].

---

We present techniques for the visual and haptic simulation of bone surgery, with a specific focus on procedures involving the temporal bone and the mandible. We discuss our approaches to graphic and haptic rendering and interactive modification of volumetric data, specifically focusing on generating force-feedback effects that are relevant to bone drilling. We then discuss how our rendering primitives and simulation architecture can be used to build surgical training techniques that are not available in traditional cadaver-based training labs, offering new possibilities for surgical education. In particular, we discuss the automatic computation of performance metrics that can provide real-time feedback about a trainee's performance in our simulator. We also present results from an experimental study evaluating the construct validity of our simulation and the validity of our performance metrics.

## 3.1 Introduction

Surgical training has traditionally revolved around an apprenticeship model: residents observe experienced surgeons in the operating room, and eventually are deemed ready to perform their first procedure [67]. In recent years, simulation-based training has emerged as a potential adjunct to this method, and the value of simulation-based learning has been more widely accepted [74]. Simulation can be a safe, cost-effective, customizable, and easily-accessible tool for gaining experience in surgery.

This section will present methods for simulating surgeries involving bone manipulation, with a specific focus on two categories of procedures: temporal bone

surgery and mandibular surgery. Section 3.1 will provide relevant clinical background on the target procedures. Section 3.2 will describe the algorithms and data structures used for interactive haptic and graphic rendering, specifically targeted toward providing key sources of intraoperative feedback for surgical interaction with bones. Section 3.3 will present the results of a study which evaluates the construct validity of our system (its ability to discriminate expert surgeons from novices). Section 3.4 will describe features of our simulation environment that do not exist in traditional, cadaver-based training labs. Section 3.5 will discuss our approach to automatically evaluating a trainee's performance in our environment.

We begin with a brief description of the relevant surgical procedures.

### 3.1.1 Temporal Bone Surgery

Several common otologic surgical procedures – including mastoidectomy, acoustic neuroma resection, and cochlear implantation – involve drilling within the temporal bone to access critical anatomy within the middle ear, inner ear, and skull base. As computer simulation is becoming a more frequently used technique in surgical training and planning, this class of procedures has emerged as a strong candidate for simulation-based learning.

The time spent on a procedure in this area is typically dominated by bone removal, which is performed with a series of burrs (rotary drill heads) of varying sizes and surface properties (Figure 3). Larger burrs are generally used for gross bone removal in the early part of a procedure, while smaller burrs are used for finer work in the vicinity of target anatomy. Surgeons employ a variety of strokes and contact techniques to precisely control bone removal while minimizing the risk of vibration and uncontrolled drill motion that could jeopardize critical structures.


**Figure 3. A typical surgical drill with an assortment of drilling burrs.**

### 3.1.2 Mandibular Surgery

Incorrect alignment of the jaws – due to congenital malformation, trauma, or disease – can result in cosmetic deformation and problems with chewing and/or breathing. Orthognathic surgeries correct such problems, typically by inducing a fracture in one or both jaws (generally using a bone saw), displacing the fractured components into an anatomically preferable configuration, and installing bone screws and/or metal plates to fix the bone segments in their new positions.

This approach is often prohibited by the severity of the deformation, the size of the separation that would be required after fracture, or the sensitivity of the surrounding soft tissue. In these cases, distraction osteogenesis is often employed as an alternative. Here a similar procedure is performed, by which only a minor separation is created intraoperatively. Instead of spanning the gap with a rigid plate, an adjustable distractor is fixed to the bone on both sides of the gap. The distractor can be used to gradually widen the fracture over a period of several weeks, allowing accommodation in the surrounding tissue and allowing the bone to heal naturally across the fracture.

These procedures are likely to benefit from surgical simulation for several reasons. The complex, patient-specific planning process and the significant anatomic variation from case to case suggests that an end-to-end simulator will assist physicians in preparing for specific cases. Furthermore, distraction procedures have been introduced to the craniofacial surgical community only within the last ten to fifteen years, and an effective simulator will significantly aid in the training and re-training of

this new class of procedures, and with the exploration of alternative techniques for effective surgeries.

### 3.1.3 Current Training Techniques

Resident training in otologic surgery typically includes dissection of preserved human temporal bones. This allows residents to become acquainted with the mechanical aspects of drilling, but does not incorporate physiological information, continuous feedback for hazard avoidance, or soft tissue work. Temporal bone labs are also costly to maintain, and cadaver specimens can be difficult to obtain in sufficient quantity. This approach also limits the precision with which an instructor can monitor a trainee's drilling performance, as the instructor cannot feel the fine details of the trainee's interaction with the bone surface, and cannot easily share the drill and bone surface for demonstration. A further limitation of cadaver-based training is that instructors have little or no mechanism for controlling anatomic variations or the presence of specific pathology that can lead to challenging training scenarios. Interactive atlases such as [79] are available for training regional anatomy. Two-dimensional simulations [26] are available for high-level procedure training.

Surgical training in craniofacial surgery typically does not include cadaver-based procedures. Most residents learn anatomy primarily from textbooks and models; surgical technique is learned through apprenticeship and procedure observation.

### 3.1.4 Previous Work

Previous work in interactive simulation of temporal bone surgery ([7], [34], [155]) has focused primarily on haptic rendering of volumetric data. Agus et al [7] have developed an analytical model of bone erosion as a function of applied drilling force and rotational velocity, which they have verified with experimental data [4]. Pflesser et al [155] model a drilling instrument as a point cloud, and use a modified version of the Voxmap-Pointshell algorithm [160] to sample the surface of the drill and generate appropriate forces at each sampled point. Each of these projects has incorporated

haptic feedback into volumetric simulation environments that make use of CT and MR data and use volume-rendering techniques for graphical display.

Agus et al [7] describe several enhancements to their simulation environment that incorporate additional skills, including the use of irrigation and suction; and additional sources of intraoperative feedback, including real-time rendering of bone dust.

Additional work has focused on non-interactive simulation of craniofacial surgery for planning and outcome prediction ([95], [101], [170]). [126] discusses preliminary work on interactive simulation of craniofacial surgery, and [65] presents a simulation architecture for arthroscopic procedures.

## 3.2 Simulation and rendering

The goal of our simulation is high-fidelity presentation of the visual and haptic cues that are present in a surgical environment. This section will discuss our overall rendering scheme, and will focus on how we present the specific cues that are relevant to surgical training.

### 3.2.1 Data Sources and Preprocessing

Models are loaded from full-head or temporal bone CT data sets, thresholded to isolate bone regions, and resampled to produce isotropic voxels, 0.5mm on a side. Using a standard resampled resolution allows us to calibrate our rendering approaches independently of the image sources used for a particular simulation case.

### 3.2.2 Hybrid Data Structure Generation

In order to leverage previous work in haptic rendering of volumetric data [153] while still maintaining the benefits of surface rendering in terms of hardware acceleration and visual effects, we maintain a hybrid data structure in which volumetric data are used for haptic rendering and traditional triangle arrays are used for graphic rendering. In order to simplify and accelerate the process of updating our polygonal data when the bone is modified, we build a new surface mesh – in which vertices correspond directly to bone voxels – rather than using the original isosurface mesh.

The voxel array representing the bone model is loaded into our simulation environment, and a polygonal surface mesh is generated to enclose the voxel grid. This is accomplished by exhaustively triangulating the voxels on the surface of the bone region, i.e.:

```
for each voxel v1
if v1 is on the bone surface
  for each of v1's neighbors v2
  if v2 is on the bone surface
    for each of v2's neighbors v3
    if v3 is on the bone surface
      generate vertices representing v1,v2,v3
      generate a triangle t(v1,v2,v3)
      orient t away from the bone surface
```

Here being 'on the bone surface' is defined as having non-zero bone density and having at least one neighbor that has no bone density. Although this generates a significant number of triangles (on the order of 200,000 for a typical full-head CT data set), we use several techniques to minimize the number of triangles that are generated and/or rendered. To avoid generating duplicate triangles, each voxel is assigned an index before tessellation, and triangles are rejected if they do not appear in sorted order. A second pass over the mesh uses the observations presented in [28] to eliminate subsurface triangles that will not be visible from outside the mesh.

Voxels are stored in a compact, in-memory hash table, which is indexed by three-dimensional grid coordinates. This allows very rapid point/volume collision-detection without excessive memory requirements.

Secondary data structures map each voxel to its corresponding vertex index, and each vertex index to the set of triangles that contain it. This allows rapid access to graphic rendering elements (vertices and triangles) given a modified bone voxel, which is critical for shading vertices based on voxel density and for re-triangulation when voxels are removed (see Section 3.2.4). Figure 4 summarizes the relevant data structures.

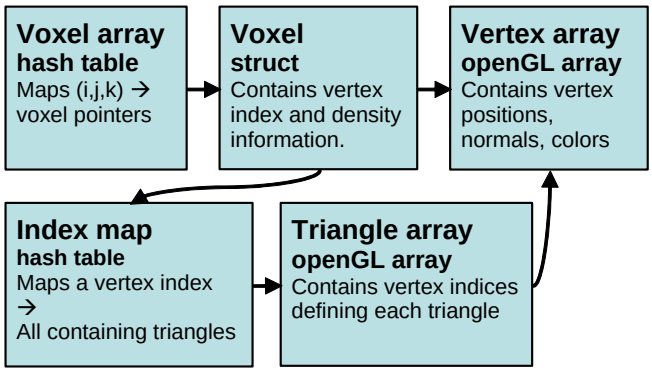



**Figure 4. A summary of the structures binding our volumetric (haptic) and surface (graphic) rendering data. When voxels are removed or modified, the corresponding vertices and triangles can be accessed from the (i,j,k) voxel index in approximately constant time.**

### 3.2.3 Haptic Rendering

Virtual instruments are controlled using a SensAble Phantom [110] haptic feedback device, which provides three-degree-of-freedom force-feedback and six-degree-of-freedom positional input. Users can select from a variety of drills, including diamond and cutting burrs ranging from one to seven millimeters in diameter. We will first discuss our approach to gross force-feedback, then we will present our methods for providing specific haptic cues that are relevant to surgical training.

#### 3.2.3.4 Gross Feedback: Volume Sampling

We initially adopted a haptic feedback approach similar to [153], in which the drill is represented as a cloud of sample points, distributed approximately uniformly around the surface of a spherical burr. At each time step, each sample point is tested for contact with bone tissue. By tracing a ray from each immersed sample point toward the center of the tool, the system can generate a contact force that acts to move that sample point out of the bone volume (Figure 5a).

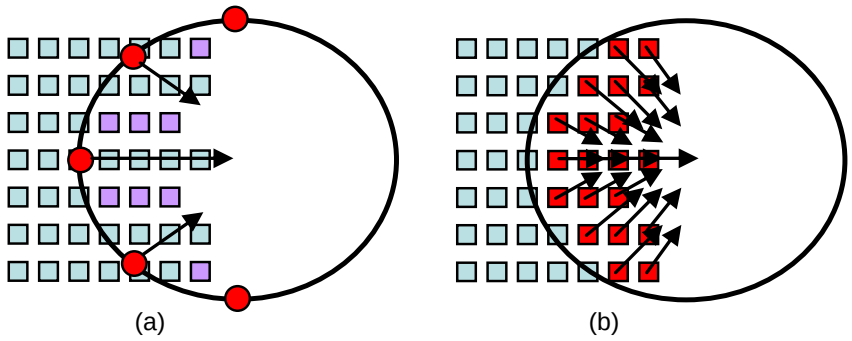



**Figure 5. Contrasting approaches to haptic rendering of drill/bone interaction. (a) The ray-tracing approach. Red points are surface samples on the surface of a spherical drill. Each sample contributes a vector to the overall force that points toward the tool center and is proportional to the penetration of the sample. Voxels labeled in purple would be missed by the raytracing algorithm, thus creating uneven bone removal. (b) Our volume-sampling approach. Here, the full volume of the drill is sampled, and each point that is found to be immersed in the bone volume contributes a vector to the overall force that points toward the center of the tool but is of unit length.**

We found that this approach worked well overall, as reported by [153], but had several undesirable artifacts. Due to sampling effects (Figure 5a), this approach produced uneven voxel removal at high resolutions, creating unrealistic bone removal patterns that depended on surface sampling. Furthermore, floating-point computations are required to find the intersection points at which rays enter and leave voxels. Since sampling density is limited by the number of samples that can be processed in a haptic timestep (approximately one millisecond), extensive floating-point computation limits the potential sampling density. This sparse sampling limits the effective stiffness of the simulation (which depends on rapid and accurate computation of penetration volume), which disrupts the illusion of contact with a highly rigid object. Furthermore, this sparse sampling limits the implementation of higher-level effects – such as bone modification that is dependent on the precise sub-parts of the drill that are used to contact the bone. These drawbacks motivate an approach that uses a higher ratio of integer to floating-point computation and allows a higher sampling density.

We thus take a more exhaustive approach to sampling the tool for haptic feedback and bone density reduction. The tool itself it discretized into a voxel grid (generally at a finer resolution than the bone grid), and a preprocessing step computes an occupancy map for the tool's voxel array. At each interactive timestep, each of the volume samples in the tool is checked for intersection with the bone volume (a constant-time, integer-based operation, using the hash table described in Section 3.2.2). A sample point that is found to lie inside a bone voxel generates a unit-length contribution to the overall haptic force vector that tends to push this sample point toward the tool center, which – with adequate stiffness – is always outside the bone volume) (Figure 5b). Thus overall penetration depth is computed based on the number of immersed sample points, rather than on the results of a per-sample ray-trace.

The overall force generated by our approach is thus oriented along a vector that is the sum of the "contributions" from individual volume sample points. The magnitude of this force increases with the number of sample points found to be immersed in the bone volume.

#### 3.2.3.5 Nonlinear magnitude computation

Because the drill is densely sampled, a large number of sample points often become immersed immediately after the drill surface penetrates the bone volume, which leads to instability during low-force contact. Reducing the overall stiffness leads to "softer" haptic feedback that does not accurately represent the stiffness of bone. We thus employ a multi-gain approach, in which the magnitude of haptic feedback is a nonlinear function of the number of immersed sample points.

More specifically, we define two gains, one of which is used when fewer than a threshold number of sample points are immersed; the other is used for deeper penetrations. This threshold is set such that the discontinuity in the force function occurs shortly after contact is initiated, so no discontinuity is perceived by the user. This relationship is summarized in Figure 6. We find that this approach allows large stiffnesses during haptic interaction, while avoiding instability during the "high-risk" period immediately following initial penetration.

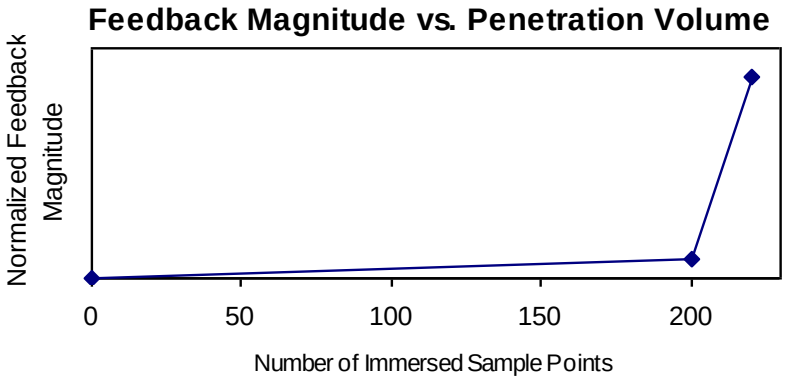



**Figure 6. Multi-gain mapping from penetration volume (number of immersed sample points) to feedback magnitude.**

Our volume-sampling approach requires sampling a significantly higher number of points than the ray-tracing approach, since the complete volume of the burr is sampled, instead of just the surface. However, the operation performed when a tool sample is found to lie within the bone volume is a constant-time computation, rather than a complex ray-tracing operation. Overall, we are able to achieve a significantly higher stiffness than they ray-tracing approach allows. We do build on the ray-tracing approach for less time-critical tasks, including bone thickness estimation (Section 3.2.9) and haptic feedback for non-physically-based tools (Section 3.2.5).

### 3.2.3.6 Modeling Drill Surface Non-uniformity

Our system also associates a "drilling power" with each sample point based on its location within the drill head; each tool voxel that intersects a bone voxel removes an amount of bone density that depends on the drilling power of the sample point. This approach allows us to simulate key aspects of drill/bone contact, particularly the fact that the equatorial surface of the burr carries a larger linear velocity than the polar surface and thus removes more bone per unit of applied force. Simulating this effect is critical for encouraging trainees to use proper drilling technique.

More precisely, the amount of bone removed per unit time by a given sample point is computed as $R_{br}$ in the following expression:

$$\theta = \text{abs}(\cos^{-1}(\mathbf{d} \bullet (\mathbf{s} - \mathbf{t}_c)))$$
$$R_{br} = f \bullet \max(0, R_{\max} - falloff \bullet \text{abs}((\pi/2) - \theta))$$

…where s is the location of this sample point, $\mathbf{t_c}$ is the location of the tool center, d is the axis of the tool handle, and θ is thus the angle between the drill handle and ($\mathbf{s}$ - $\mathbf{t_c}$). The expression abs($\pi/2$ – θ) is thus the "latitude" of the current sample point. *falloff* is a constant parameterizing the non-uniformity of the drill surface. If falloff is zero, the pole and the equator of the drill remove bone with equal efficiency. $R_{max}$ is the maximum rate of bone removal per unit force, and *f* is the magnitude of force currently being applied by the user. The computation of latitude is summarized in Figure 7. Note that falloff parameters are precomputed for drill samples to avoid performing expensive arc-cosine operations hundreds of times per haptic timestep.

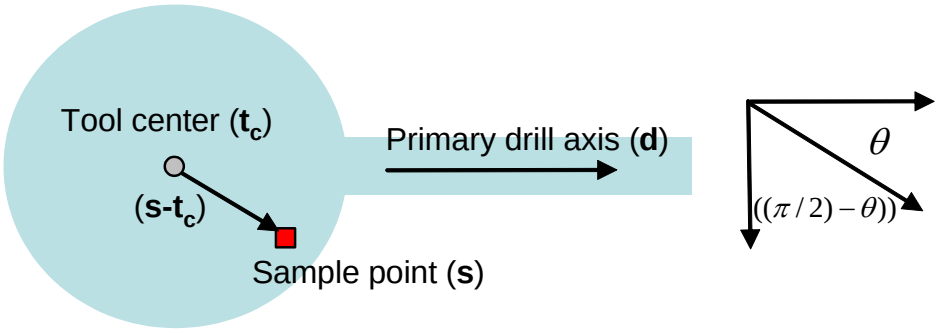



**Figure 7. The computation of the "latitude" of a volume sample point for bone removal rate computation.**

This approach allows us to encourage proper drilling technique and to model critical differences among burr types. For example, our model captures the fact that cutting burrs typically show more dependence on drilling angle than diamond burrs do, but have higher overall bone removal rates. A cutting burr would thus be associated with both a higher $R_{max}$ and a higher falloff in the above expression.

### 3.2.3.7 Modeling Tangential Forces

Another property of surgical drills that should be accurately represented in a simulation environment is their tendency to drag the user along the surface of the bone, due to the contact forces between the teeth of the drilling burr and the bone (Figure 8). Stroking the drill on the bone surface in a direction that allows these forces to *oppose* a surgeon's hand motion permits the surgeon to control the velocity of the drill. Stroking the drill such that these forces complement the surgeon's hand motion causes the drill to catch its teeth on the bone and rapidly "run" in the direction of movement, which can be extremely dangerous. Simulating this effect is thus critical to training correct drilling technique.

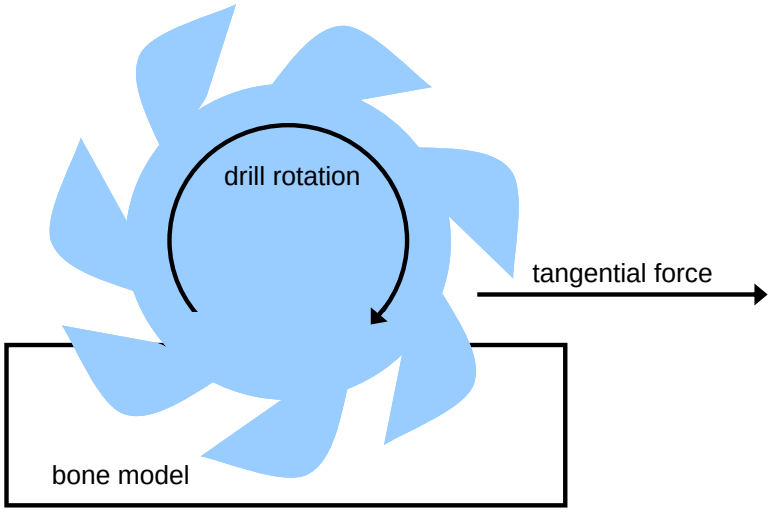



**Figure 8. A spinning, burred drill creates a tangential force that propels the drill along the bone surface.**

Modeling the contact forces between the individual teeth in the drill's geometry and the bone surface would be computationally expensive, so we again employ our dense sampling approach to approximate tangential drill forces during the computation of penalty forces.

Each sample that is found to be immersed in the bone (i.e. the red samples in Figure 5a) computes its own tangential force vector, according to:

$$f_{\tan} = (p - sc) \times d$$

…where $\mathbf{f_{tan}}$ is the tangential force created by this sample, **p** is the position of this sample, **sc** is the center of the "slice" of the drill in which this sample lies (the sample position projected onto the drill axis), and **d** is the primary axis of the drill (and thus the axis of rotation), as shown in Figure 7.

The vector **(p – sc)** is a vector from the tool axis to this sample point, an approximation of the local surface normal (the true surface normal is generally unknown, since most samples are not on the surface of the model and thus don't have defined normals). The drill axis vector is normalized to unit length, and the magnitude of the vector **(p – sc)** indicates its distance from the tool axis and thus its linear velocity (since the drill spins at constant rotational velocity, samples farther from the axis of rotation carry larger linear velocity than close near the axis of rotation). The cross-product **(p – sc) × d** is thus scaled according to sample velocity, and is perpendicular to both the drill's axis and the approximate surface normal.

Summing these vectors over all samples that are found to lie on the bone creates a net force that simulates the interaction between the teeth of the drill and the bone surface. Scaling this vector by -1 is equivalent to reversing the "handedness" of the drill.

### 3.2.3.8 Modeling Drill Vibration using Recorded Data

Another key aspect of the haptic sensation associated with drilling is the vibration of the instrument, which varies with applied force and with burr type. In order to generate realistic drill vibration frequencies, we outfitted a physical drill with an accelerometer and collected vibration data at a variety of applied drilling forces. These data are summarized in Figure 9. The key spectral peaks were identified for each burr type and used to synthesize vibrations during the simulation. Since we are driving our haptic feedback device at approximately 1.5 kHz, we are unable to preserve the highest-frequency vibrations identified in these experimental recordings. However, we are able to preserve the lower-frequency harmonics and the variations in vibration associated with changes in burr type and/or changes in applied drilling force.

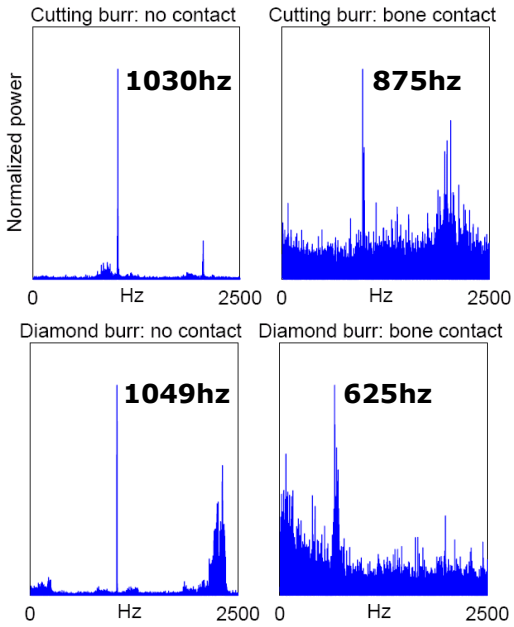



**Figure 9. A spectral representation of drill vibration, collected from cutting (top row) and diamond (bottom row) drilling burrs, when in contact with bone and when powered but held away from the bone surface. The frequencies of the largest peaks are highlighted. The sharp spectral peaks make this data suitable for real-time vibration synthesis.**

### 3.2.4 Data Manipulation

When bone voxels are removed from our environment, our hybrid data structure requires that the area around the removed bone be retessellated. Consequently, bone voxels are queued by our haptic rendering thread as they are removed, and the graphic rendering thread retessellates the region around each voxel pulled from this queue. That is, for each removed voxel, we see which of its neighbors have been "revealed" and create triangles that contain the centers of these new voxels as vertices. Specifically, for each removed voxel v, we perform the following steps:

```
for each voxel v' that is adjacent to v
  if v' is on the bone surface
    if a vertex has not already been created
    to represented v'
```

```
      create a vertex representing v′
      compute the surface gradient at v′
      queue v′ for triangle creation

for each queued voxel v′
  generate triangles adjacent to v′ (see below)
```

Once again, a voxel that is "on the bone surface" has a non-zero bone density and has at least one neighboring voxel that contains no bone density. When all local voxels have been tested for visibility (i.e. when the first loop is complete in the above pseudocode), all new vertices are fed to a triangle generation routine. This routine finds new triangles that can be constructed from new vertices and their neighbors, orients those triangles to match the vertices' surface normals, and copies visible triangles to the "visible triangle array" (see Section 3.2.7). The reason for "queuing triangles for triangle creation" is that the generation of triangles – performed in the second loop above – depends on knowing which local voxels are visible, which is only known after the completion of the first loop.

### 3.2.5 Additional Tools

An additional bone modification tool allows the introduction of large bone cuts via a planar cut tool (see Figure 10). This tool generates no haptic feedback and is not intended to replicate a physical tool. Rather, it addresses the need of advanced users to make rapid cuts for demonstration or for the creation of training scenarios. Bone removal with this tool is implemented by discretizing the planar area – controlled in six degrees of freedom – into voxel-sized sample areas, and tracing a ray a small distance from each sample along the normal to the plane. This is similar to the approach used in [153] for haptic rendering, but no haptic feedback is generated, and each ray is given infinite "drilling power", i.e. all density is removed from any voxels through which each ray passes. The distance traced along each ray is controlled by the user. This allows the user to remove a planar or box-shaped region of bone density, demonstrated in Figure 10b. This approach will often generate isolated fragments of bone that the user wishes to move or delete. This operation is discussed in Section 3.2.6.

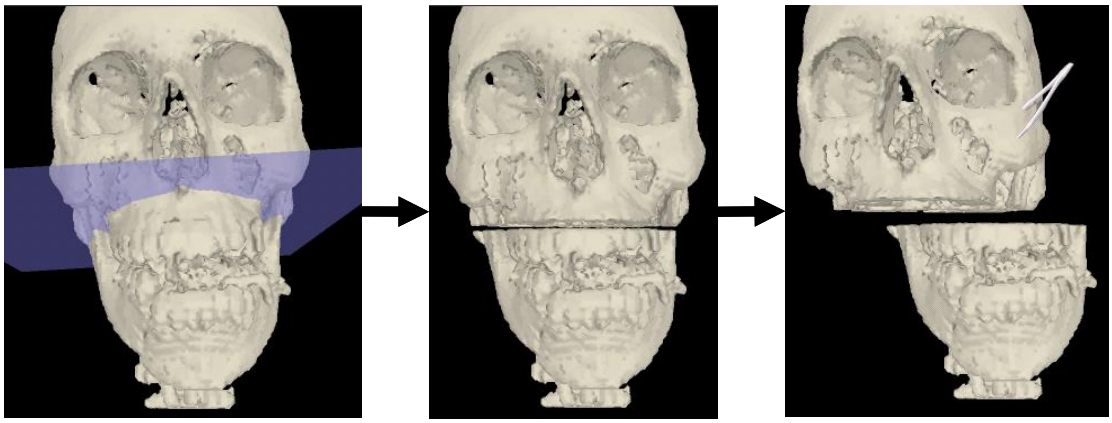


**Figure 10. The use of the cut-plane tool and the independent manipulation of discontinuous bone regions. (a) The cut-plane tool is used to geometrically specify a set of voxels to remove. (b) The volume after voxel removal. (c) The flood-filling thread has recognized the discontinuity, and the bone segments can now be manipulated independently.**

A final set of tools allows the user to manipulate rigid models that can be bound to bone objects. This is particularly relevant for the target craniofacial procedures, which center around rigidly affixing metal plates to the patient's anatomy. We thus provide models of several distractors and/or industry-standard bone plates (it is straightforward to add additional models). The inclusion of these plate models allows users to plan and practice plate-insertion operations interactively, using industry-standard plates. Collision detection for haptic feedback is performed using a set of sample points, as was the case with drilling tools. In this case, the sample points are generated by sampling 100 vertices of each model and extruding them slightly along their normals (because these models tend to be very thin relative to our voxel dimensions) (Figure 11a). For this tool/bone contact, which generally involves objects with much larger volumes than the drill tools, we elected to use the ray-tracing approach presented in [153]. This approach allows reasonable haptic feedback with lower numbers of samples than the volumetric approach we use for our drilling tools (Section 3.2.3). Since there is no well-defined tool center toward which we can trace rays for penetration calculation, rays are traced along the model's surface normal at

each sample point. At any time, the user can rigidly affix a plate tool to a bone object with which it is in contact using a button on the haptic device (Figure 11b,c,d).

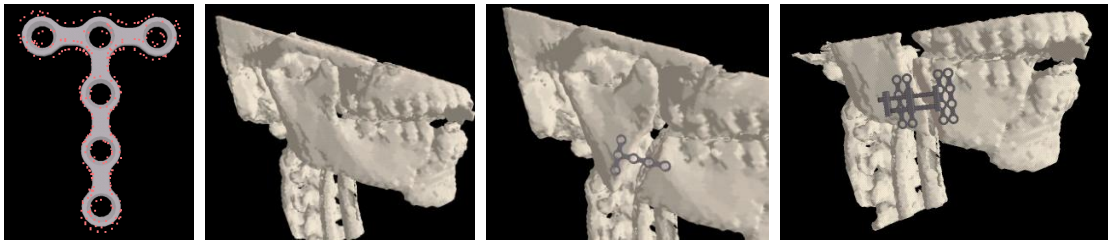



**Figure 11. The modeling and attachment of rigid bone plates. (a) The surface of a bone plate after sampling and extrusion. (b) A bone surface before modification. (c) The same bone surface after drilling, distraction, and plate attachment. (d) The same bone surface after drilling, distraction, and distractor insertion.**

### 3.2.6 Discontinuity Detection

A critical step in simulating craniofacial procedures is the detection of cuts in the bone volume that separate one region of bone from another, thus allowing independent rigid transformations to be applied to the isolated bone segments.

In our environment, a background thread performs a repeated flood-filling operation on each bone structure. A random voxel is selected as a seed point for each bone object, and flood-filling proceeds through all voxel neighbors that currently contain bone density. Each voxel maintains a flag indicating whether or not it has been reached by the flood-filling operation; at the end of a filling pass, all unmarked voxels (which must have become separated from the seed point) are collected and moved into a new bone object, along with their corresponding data in the vertex and triangle arrays. Figure 12 summarizes this operation and provides an example.

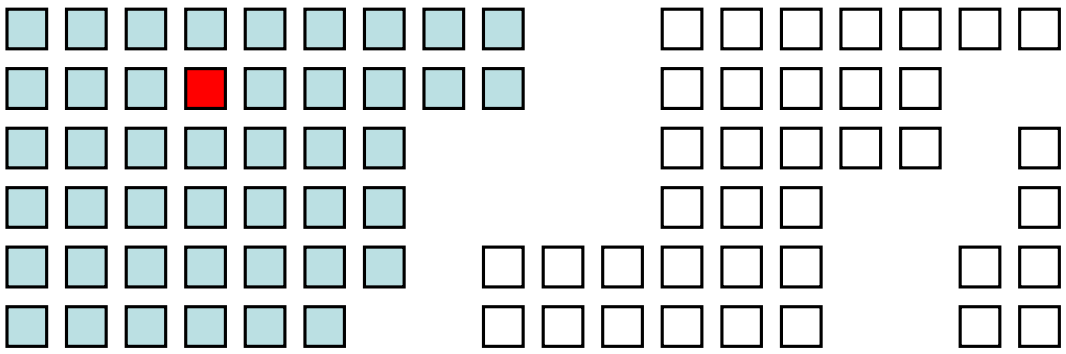


**Figure 12. Discontinuity detection by flood-filling. The seed voxel is highlighted in red, and the shaded (blue) voxels were reached by a flood-filling operation beginning at the seed voxel. These voxels thus continue to be part of the same bone object as the seed voxel, while the unshaded voxels on the right have become disconnected and thus are used to create a new bone object. In a subsequent pass through the flood-filling algorithm, a third bone object would be created, because the unfilled voxels are further fragmented.**

Figure 10a and Figure 10c display a bone object that has been cut and the subsequent independent movement of the two resulting structures. Here – for demonstration – the cut-plane tool is used to create the fracture; during simulated procedures, fractures are generally created by the drilling/sawing tools.

### 3.2.7 Graphic Rendering

To take advantage of the fact that the user does not frequently change the simulation's viewing perspective, we maintain two triangle arrays, one containing the complete tessellation of the current bone volume (the "complete array"), and one containing only those that are visible from positions close to the current camera position (the "visible array"). The latter array is initialized at startup and is re-initialized any time the camera comes to rest after a period of movement. Visible triangles are those with at least one vertex whose normal points towards (less than 90 degrees away from) the camera. Because this visibility-testing pass is time-consuming, it is performed in the background; the complete array is used for rendering the scene during periods of camera movement (when the visible array is considered 'dirty') and during the reinitialization of the 'visible' array.

As an additional optimization, we use the nvtristrip library [143] to reorder our triangle and vertex arrays for optimal rendering performance. We could have further reduced rendering time by generating triangle strips from our triangle lists, but this would add significant computational complexity to the process of updating the surface mesh to reflect changes to the underlying voxel grid.

### 3.2.8 Bone Dust Simulation

We also build on the work presented in [7] to provide a simulation of bone dust accumulation, which is particularly critical in otologic procedures. Bone dust tends to accumulate in the drilling area, and must be suctioned off to enhance visibility of the bone surface.

Agus et al [7] simulate the behavior of individual particles of bone dust, sampling a subset of the particles in each rendering pass to minimize the computational load demanded by the simulation. Since individual particles of bone dust are not generally visible, it is unnecessary to simulate particulate motion. Therefore we take an Eulerian approach similar to [186], in which we discretize the working region into a three-dimensional hashed grid. Rather than tracking individual particles, we track the density of particles contained in each grid cell. This allows us to simulate the piling of dust particles, particle flow due to gravity, and particle movement due to tool contact for all accumulated bone dust, without simulating individual particles. Gravity and tool forces transfer density between neighboring grid cells, rather than modifying the velocity of individual particles.

Each grid cell containing bone dust is rendered as partially-transparent OpenGL quad, whose dimensions scale with the density of dust contained in that cell. This provides a convincing representation of accumulated particle volume and density, and does not require that we render each particle (that is, each quantum of density) individually.

This grid-based approach significantly reduces computation and rendering time relative to a particle-based (Lagrangian) approach. Coupled with the hash table we use to minimize memory consumption for the grid, we are able to render large quantities of accumulated bone dust without impacting the interactive performance of

the application. Figure 13 shows a volume of accumulated bone dust and the suction device used by the trainee to remove it. The suction device is controlled with an additional Phantom haptic interface.

**Figure 13. Bone dust simulation. The user has removed a volume of bone, which has now accumulated as bone dust. The physical simulation has allowed the bone dust to fall to the bottom of the drilled area. The user is preparing to remove the bone dust with the suction device.**

### 3.2.9 Data-Driven Sound Synthesis

Sound is a key source of intraoperative feedback, as it provides information about drill contact and about the nature of the underlying bone. We simulate the sound of the virtual burr as a series of noisy harmonics, whose frequency modulates with applied drilling force. Building upon the harmonic-based synthesis approach presented in [34], we have recorded audio data from cutting and diamond drill burrs applied to cadaver temporal bone a under a series of drilling forces, in order to determine the appropriate frequencies for synthesized sound, as well as the dependence of this data

on drill type and applied drilling force. Figure 14 summarizes the spectral information collected from diamond and cutting burrs.

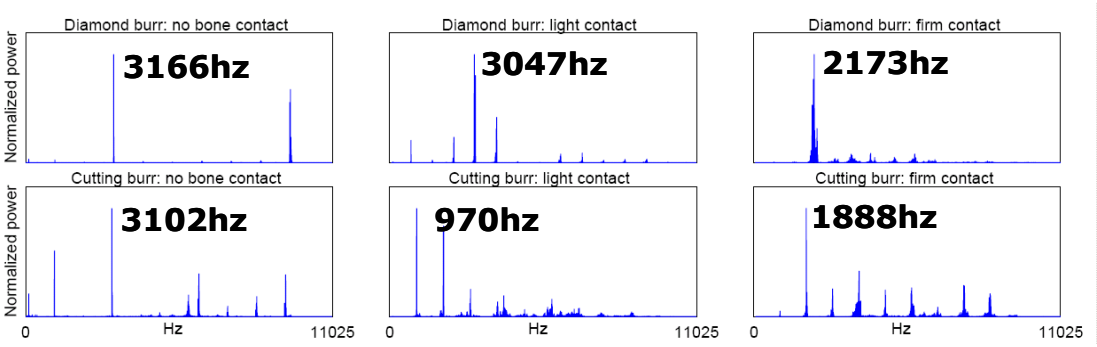



**Figure 14. A spectral representation of audio data collected from diamond (top row) and cutting (bottom row) drilling burrs. Columns represent no bone contact, bone contact without significant pressure, and bone contact with a typical drilling pressure (applied by an experienced surgeon). The sharp spectral peaks and distinct variation among drill types and contact forces make this data suitable for real-time synthesis.**

Sound can also be a key indicator of bone thickness intraoperatively; sound quality and frequency change significantly as the drill contacts a thin layer of bone, providing a warning that the surgeon is approaching sensitive tissue. In our simulator, the pitch of the synthesized sound increases when the drilled area becomes thin. In order to estimate the thickness of bone regions, we used a raytracing algorithm similar to that used for haptic rendering in [153]. At each voxel that is determined to be on the surface of the bone, the surface gradient is used to approximate the surface normal, and a ray is cast into the bone along this normal. The ray is traced until it emerges from the bone volume, and the thickness is estimated as the distance from the ray's entry point to its exit point. For sound synthesis, this thickness is averaged over all surface voxels with which the drill is in contact. Below an empirically selected thickness threshold, sound frequency increases linearly with decreasing bone thickness. The slope of this relationship is selected so that the key harmonics span the same range of frequencies in simulation that they do in our measured data.

## 3.3 Results: Construct Validity

### 3.3.1 Experimental Procedure

The surgical simulation community defines several levels of "validity" – the ability for a simulator to mimic the real-world properties of the environment it aims to represent. The present study assesses the "construct validity" of our simulation environment: the ability to explain subject behavior in simulation with appropriate parameters describing subject experience level. In other words, expert surgeons should perform objectively better on a simulated surgical task than novices.

For the present study, fifteen right-handed participants were asked to perform a mastoidectomy (removal of a portion of the temporal bone and exposure of relevant anatomy) in our simulator. Participants included four experienced surgeons, four residents in head and neck surgery with surgical experience, and seven novices with no surgical experience.

Participants were presented with a tutorial of the simulator and were given fifteen minutes to practice using the haptic devices and the simulator's user interface. Participants were then presented with an instructional video describing the target procedure, and were given access – before and during the procedure – to still images indicating the desired appearance of the bone model at various stages in the procedure (Figure 15). Participants were asked to perform the same procedure twice.

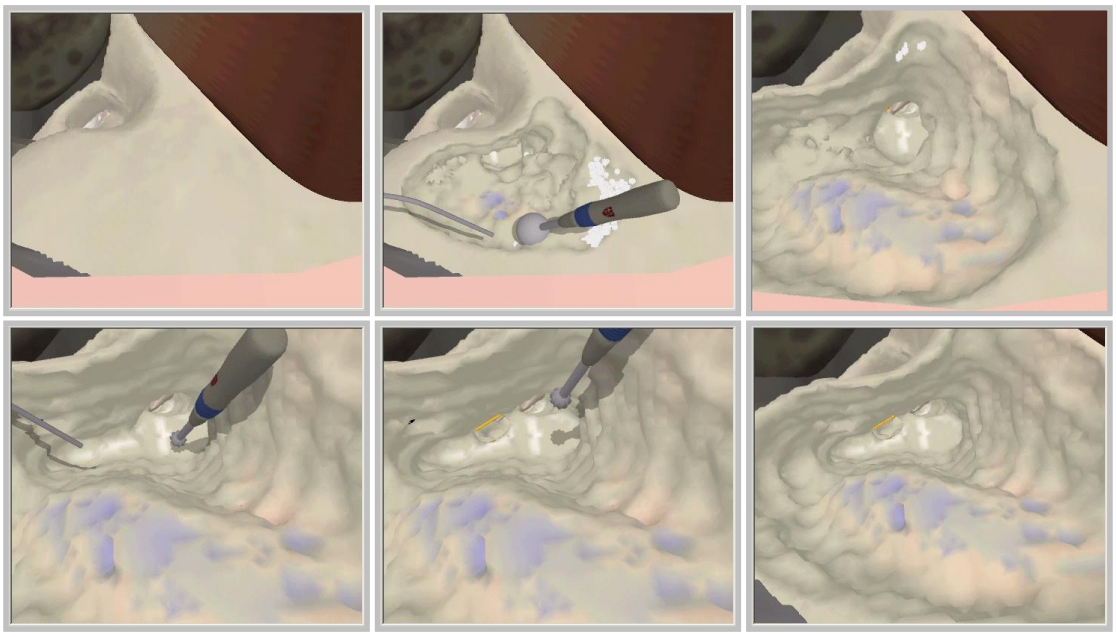


**Figure 15. Still images presented to experimental participants, indicating the stages of the mastoidectomy procedure.**

Each participant's hand movements, haptic forces, and surgical interactions were logged to disk then later rendered to video. Videos were scored on a scale of 1 to 5 by an experienced head and neck surgery instructor; the instructor was not aware of which videos came from which subjects and viewed them in randomized order. This scoring approach is similar to the approach used to evaluate resident progress in a cadaver training lab. Our hypothesis is that participants with surgical experience should receive consistently higher scores than those with no surgical experience.

Figure 16 shows a summary of the experimental results. Participants with surgical experience received a mean score of 4.06, and novices received a mean score of 2.31, a statistically significant difference according to a one-tailed t-test ($p < 0.0001$). This clear difference in performance when operating in our simulator demonstrates the construct validity of the system.

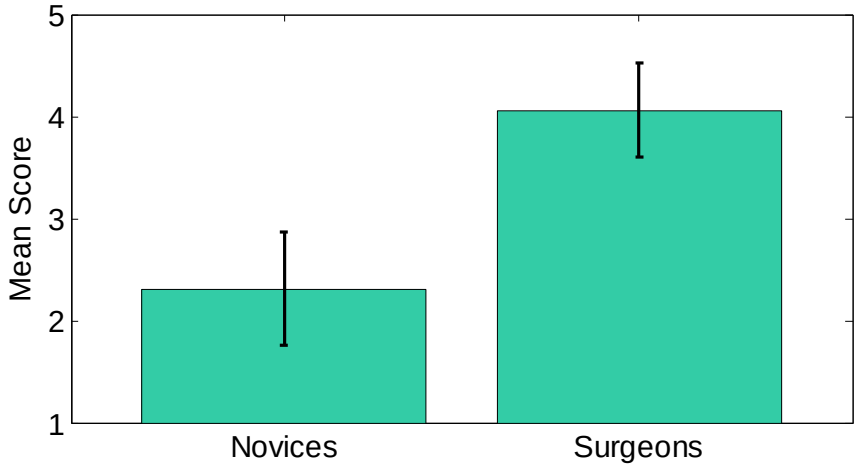



**Figure 16. Mean scores for simulated mastoidectomies performed by novice participants (left) and participants with surgical experience (right). Error bars indicate 95% confidence intervals.**

## 3.4 Novel Training Techniques

The previous subsections of this section discussed our simulator's approach to replicating interaction with bones, i.e. replicating the features available in a traditional cadaver-based training lab. The following section discusses our incorporation of training features that are not possible in a traditional training lab, and thus demonstrate the potential for simulation to not only replicate but also extend existing training techniques.

### 3.4.1 Haptic Tutoring

Surgical training is typically focused on visual observation of experienced surgeons and verbal descriptions of proper technique; it is impossible for a surgeon to physically demonstrate the correct 'feel' of bone manipulation with physical tools. With that in mind, we have incorporated a 'haptic tutoring' module into our environment, allowing a trainee to experience forces that are the result of a remote user's interaction with the bone model.

Ideally, the trainee would experience both the movements of the instructor's tool and the force applied to/by the instructor, but it is difficult to control both the position and the force at a haptic end-effector without any control of the compliance of the user's hand. To address this issue, we bind the position of the trainee's tool to that of an instructor's tool (running on a remote machine) via a low-gain spring, and add the resulting forces to a 'playback' of the forces generated at the instructor's tool. I.e.:

$$F_{trainee} = K_p(P_{trainee} - P_{instructor}) + F_{instructor}$$

…where $F_{instructor}$ and $F_{trainee}$ are the forces applied to the instructor's and trainee's tools, and $P_{instructor}$ and $P_{trainee}$ are the position of the instructor's and trainee's tools. $K_p$ is small enough that it does not interfere significantly with the perception of the high-frequency components transferred from the instructor's tool to the trainee's tool, but large enough that the trainee's tool stays in the vicinity of the instructor's tool. In practice, the error in this low-gain position controller is still within reasonable visual bounds, and the trainee perceives that he is experiencing the same force and position trajectory as the instructor.

We use the same approach and the same force constants for "haptic playback", allowing a user to play back force data collected from a previous user's run through our system. This has potential value both for allowing trainees to experience the precise forces applied during a canonically correct procedure, and for allowing an instructor to experience and evaluate the precise forces generate during a trial run by a trainee.

### 3.4.2 Neurophysiology Console Simulation

Another goal of our simulation environment is to train the surgical skills required to avoid critical and/or sensitive structures when using potentially dangerous tools. The inferior alveolar nerve is at particular risk during most of the procedures this environment is targeting. We thus incorporate a virtual nerve monitor that presents the user with a representation of the activity of nerve bundles in the vicinity of the

procedure (Figure 17a). Nerves are currently placed explicitly for training scenarios; future work will include automatic segmentation of large nerves from image data.

This approach will also potentially contribute to the simulation-based training of a complete surgical team, which typically involves several technicians focused on neurophysiology monitoring. Simulated neural data is streamed out via Ethernet for remote monitoring, and can be visualized on a console that is similar to what would be available intraoperatively to a technician. Our system uses the visualization and analysis software distributed with the Cerebus neural recording system (CyberKinetics, Inc.) (Figure 17b).

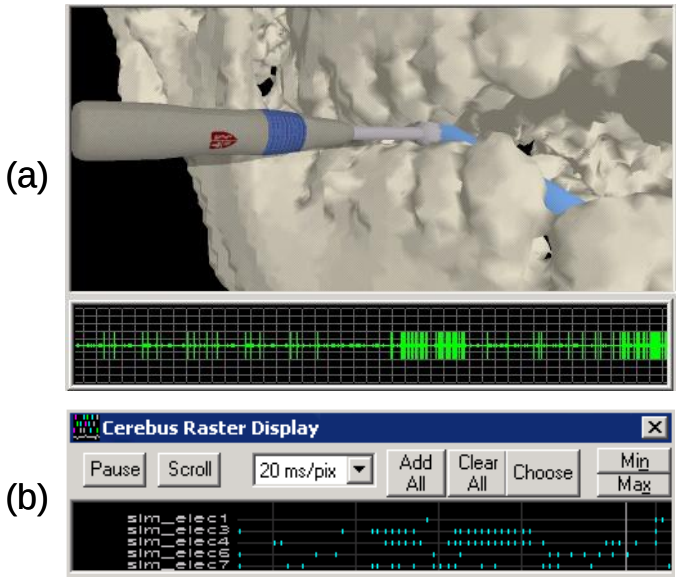



**Figure 17. Virtual neurophysiology monitoring. (a) The user drills near a simulated nerve (in blue) and views a real-time simulated neural monitor, which also provides auditory feedback. (b) A remote user visualizes the activity of several simulated nerves, observing activity bursts when the user approaches the nerve structures.**

## 3.5 Automated Evaluation and Feedback

Another exciting possibility for virtual surgery is the use of simulation environments to automatically evaluate a trainee's progress and provide targeted feedback to help improve a user's surgical technique.

A straightforward approach to evaluating a trainee's performance on the simulator is determining whether a given objective has been achieved while avoiding injury to vulnerable structures (such as nerves, ossicles, or veins). However, many of the finer points of technique that surgeons learn are taught not because failure to adhere to them will *necessarily* result in injury, but because it increases the *likelihood* of injury. Therefore, it is useful to be able to quantify the risk inherent in the trainee's performance.

This section describes several metrics for evaluating a user's bone-drilling technique, and our approach to visualizing these metrics. We also present approaches to validating these metrics (confirming that they are medically meaningful) and initial validation results.

### 3.5.1 Visibility Testing

One of the most important ways in which risk is minimized in temporal bone surgery is by taking care to only remove bone that is within the line of sight, using a "saucerizing" drilling technique (removing bone so as to create a saucer-shaped cavity on the bone surface). This enables the surgeon to avoid vulnerable structures just below the bone surface, using subtle visual cues that indicate their locations. If instead some bone is removed by "undercutting" (drilling beneath a shelf of bone that obscures visibility), there is increased risk of structure damage.

In our environment, as each voxel of bone is removed, the simulator determines whether this voxel was visible to the user at the time of removal. Making use of the same ray-tracing techniques that are used for haptic rendering (Section 3.2.5), a line is traced from the removed voxel to the virtual eye point. If any voxels (other than those currently in contact with the drill) are intersected by this ray, the removed voxel is determined to be invisible.

During or after a virtual procedure, a user can visualize the visibility/invisibility of every voxel he removed, to explore the overall safety of his technique and find specific problem areas. Voxels that were visible when removed are shown in one color while those that were obscured are rendered in another color (Figure 18). The scene may also be rotated and rendered with only selected structures

visible, allowing unobstructed visualization of the locations of the removed voxels and their proximities to crucial structures.

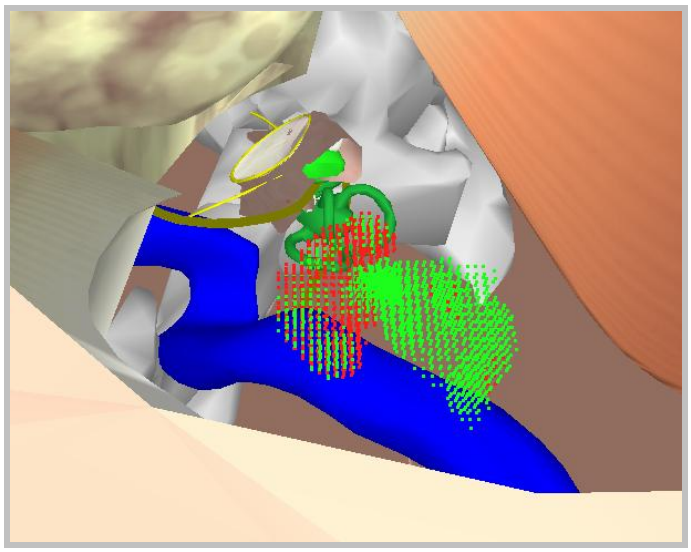


**Figure 18. Visualization of removed voxel visibility. In this simulation, the trainee has correctly "saucerized" on the right side, removing only visible bone, while he "undercut" on the left side, removing bone that was hidden by other bone. This interactive visualization – in which the bone itself is not rendered – displays the regions in which he exercised proper technique (visible voxels in green) and regions in which he did not (obscured voxels in red). Undercutting in close proximity to the sigmoid sinus (in blue) was dangerous as he could not see the visual cues indicating the vein's location below the bone surface.**

Although it makes intuitive sense that voxel visibility should be an appropriate metric for evaluating a user's performance, it is important to validate this metric – and all automatic metrics – against a clinically-standard assessment of user performance. In this case, we use the data collected from the user study presented in Section 3.3, which includes complete simulated procedures by experts and novices, along with scores assigned to each simulated procedure by an experienced surgical instructor. A metric that correlates well to an instructor's manually-assigned scores is likely to be an effective metric for automatic user evaluation.

Figure 19 shows the results of correlating computed voxel visibilities to an instructor's score (on a scale of 1 to 5) for each simulated procedure performed by

each of our study participants. Linear regression shows a correlation coefficient of 0.68, which is particularly high considering that the manual evaluation was based on a wide array of factors, only one of which was voxel visibility. This approach is suitable for assessing the effectiveness of individual metrics, which can be combined to form an overall score for a simulated procedure.

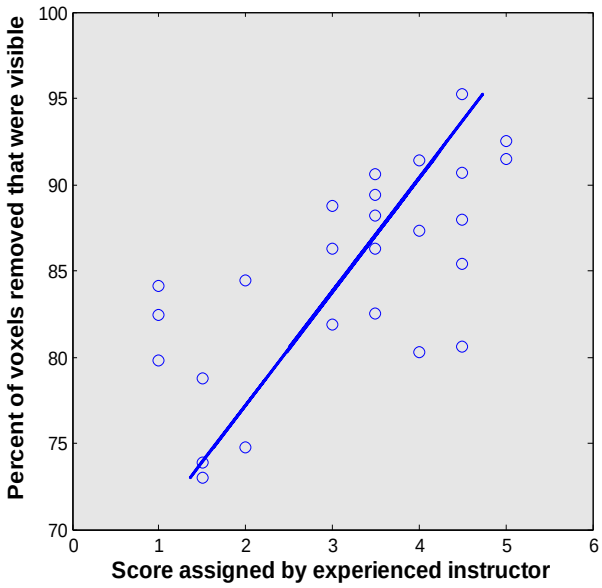



**Figure 19. Relationship between expert-assigned scores (x axis) and computed voxel visibility (y-axis), along with a linear fit (R=0.68, p<0.001). Each dot represents one pass through the simulated procedure by one subject. The strong correlation supports the value of computed visibility as an automatic performance metric.**

### 3.5.2 Learning Safe Forces

Another component of safe drilling is applying appropriate forces and operating the drill at appropriate speeds. The acceptable range of forces and speeds is closely related to the drill's distance from vulnerable structures. However, this function is difficult for a human, even an expert surgeon, to precisely quantify. Therefore, we learn maximal safe forces and speeds via statistical analysis of forces, velocities, and distances recorded during a run of the simulation by experienced surgeons. Trainees'

performance can then be compared to the experts' values, and areas in which excessive speeds or forces were applied can be visualized and presented to the user.

For example, Figure 20 shows the force profiles of all expert and novice study participants as they approached a critical and sensitive structure, the chorda tympani, a branch of the facial nerve. At the instant that any voxel within 3cm of this structure was removed, the user's applied force was recorded. These samples were then sorted by distance from the nerve and binned into 0.2cm intervals; the mean value of each bin was computed and plotted in Figure 20. The profiles for experts and novices are significantly different, as indicated by the plotted confidence intervals. Experts clearly tend to use lower forces overall in the vicinity of this critical structure, and reduce their forces as they approach, a trend not seen in the novice plots.

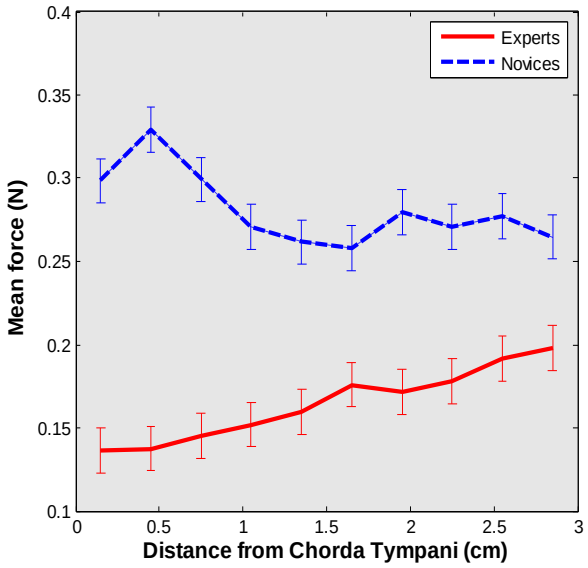



**Figure 20. Forces applied by experts and novices in the vicinity of the chorda tympani (a sensitive branch of the facial nerve). Error bars indicate 95% confidence intervals. Experts display a significantly different force profile in this region than novices, as experts tend to reduce their applied forces when approaching the nerve.**

### 3.5.3 Learning Correct Bone Regions for Removal

In addition to instantaneous metrics like force and visibility, an instructor evaluating a surgical trainee would also evaluate the overall shape of the drilled region after a complete procedure, i.e. the set of voxels removed by the trainee.

To capture this important criterion in a quantitative metric, we use a Naïve Bayes approach to categorize "correct" and "incorrect" drilling regions. We assume that voxels from the full voxel mesh are chosen for removal (drilling) according to separate distributions for experts and novices. For each voxel, we compute the probability that an expert would remove this voxel and the probability that a novice would remove this voxel. Then for each subject's run through a simulated procedure, we look at the set of removed voxels and ask "what was the probability that an expert (or novice) performed this procedure?", by multiplying together the probabilities of each removed voxel. We then compute the ratio of these cumulative probabilities ( $p_{expert}$ and $p_{novice}$ ) and take the log of that ratio, to compute a scalar value that estimates the correctness of the drilled region ( $\log(p_{expert}/p_{novice})$ ).

We would like to show that this is a valid performance metric by correlating it with scores assigned by an experienced instructor, as we did in Section 3.5.1. Figure 21 shows the result of this analysis, along with a linear regression onto the scores assigned by an instructor (R=0.76). Again, the high correlation suggests that this is a valuable component in a suite of individual metrics than can produce an accurate estimate of trainee performance.

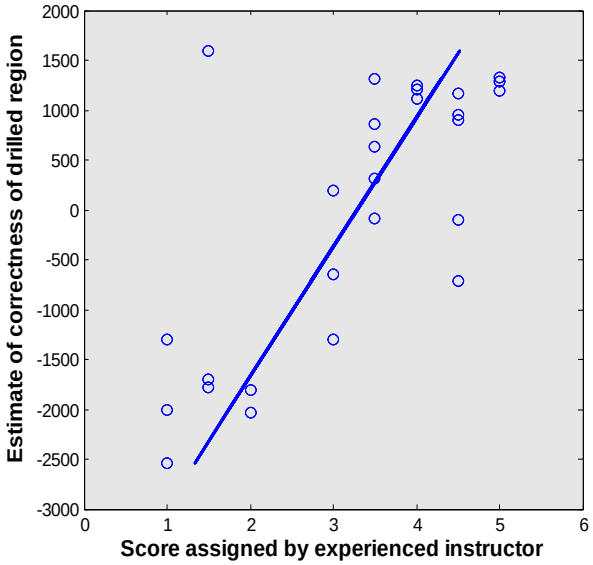



**Figure 21. Relationship between expert-assigned scores (x axis) and estimate of drilled region correctness (y-axis), along with a linear fit (R=0.76, p<0.001). Each dot represents one pass through the simulated procedure by one subject. The strong correlation supports the validity of our drilled-region-correctness estimate as an automatic performance metric.**

## 3.6 Conclusion and Future Work

We have described a system for visuohaptic simulation of bone surgery, including a volume-sampling algorithm for haptic rendering and a hybrid data structure for linking visual and haptic representations of volume data. We presented empirical results evaluating the construct validity of our system, and we presented our approach to building task-level scenarios and evaluation mechanisms on top of our physical simulation.

Subsequent work on the simulation environment will focus on incorporating a representation of soft tissue simulation into our environment, to enable the representation of more complete procedures, including, for example, skin incision and tumor resection. Subsequent work on our automated evaluation techniques will focus

on the development of additional automated metrics and the visualization of automated metrics.

Supplemental material for this section, including movies and images of the simulation environment, is available at:

http://cs.stanford.edu/~dmorris/projects/bonesim/

# 4 Haptic Training Enhances Force Skill Learning

Traditional haptic rendering techniques focus primarily on *simulation*: a virtual environment uses a haptic device to replicate a sensation experienced in the physical world. This is the goal, for example, of the haptic rendering techniques described in Section 3. This approach has been used in most haptic simulation environments oriented toward skill training. This paradigm is analogous to practicing any skill in the physical world: a skill is performed repetitively under realistic conditions to improve a trainee's ability to perform that skill in the future.

An alternative paradigm uses a haptic device to present forces that do not represent a "realistic" interaction with a simulated environment. The possibility of generating haptic forces other than physical interaction forces appeared intriguing during the development of the simulation environment described in Section 3. In particular, haptic feedback offers the possibility of *demonstrating* manual skills, with the user passively receiving information via the haptic device. However, it was not obvious that force-sensitive skills could be learned in this manner, so we chose to create an abstract task that would allow us to evaluate the potential for haptic feedback to teach force-sensitive motor skills. This task is the topic of Section 4 of this dissertation.

The surgical skills required for bone drilling are sensitive to both movement (position) and force, and are guided by both visual landmarks and force feedback. This section thus presents an experiment in which subjects are taught visually-guided

patterns of forces and are asked to recall those forces. The results indicate that this form of training – which we refer to as "haptic mentoring" – can indeed augment visual training for the class of skills we examined.

Related techniques have been implemented in our simulation environment. The simulator guides a user through the surgical field, displaying "correct" interaction forces (those experienced previously by a trained surgeon). Data can be played back from a file or streamed in real-time from a surgeon using our environment (Section 3.4.1) (Figure 1). Future work will assess the utility of this feature in the context of surgical training.

The worked presented here has been submitted as [129].

---

This section explores the use of haptic feedback to teach an abstract motor skill that requires recalling a sequence of forces. Participants are guided along a trajectory and are asked to learn a sequence of one-dimensional forces via three paradigms: haptic training, visual training, or combined visuohaptic training. The extent of learning is measured by accuracy of force recall. We find that recall following visuohaptic training is significantly more accurate than recall following visual or haptic training alone. This suggests that in conjunction with visual feedback, haptic training may be an effective tool for teaching sensorimotor skills that have a force-sensitive component to them, such as surgery. We also present a dynamic programming paradigm to align and compare spatiotemporal haptic trajectories.

## 4.1 Introduction

Haptic feedback has become an integral component of numerous simulation systems, particularly systems designed for teaching surgical skills (e.g. [18], [128], [209]). Haptic rendering in nearly all such simulation environments has been designed to realistically replicate the real-world forces relevant to a particular task. Recent results

suggest that simulation environments can contribute to users' learning of real motor skills [215] and to users' perception of virtual object shapes [92]. In contrast, Adams et al [3] found no significant learning benefit from haptic feedback for a manual assembly task, despite an overall benefit from training in a virtual environment.

Although haptic feedback is often used to replicate real-world interaction forces, haptics has the potential to provide cues that are not available in the physical world. In particular, haptic feedback can be used as a channel for presenting motor patterns that a user is expected to internalize and later recall. Feygin et al [58] refer to this approach as "haptic guidance", and found that haptic feedback contributes to learning spatiotemporal trajectories. Williams et al [211] employed this technique in a medical simulator and also found that it contributed to learning position trajectories. Patton and Mussa-Ivaldi [150] employ an implicit version of this technique, allowing users to adapt to a movement perturbation in order to teach a motion that is opposite to the perturbation. In contrast, Gillespie et al [66] used a similar approach to teach a motor control skill, and found no significant benefit from haptic training, although haptic training did affect the strategy that participants used when performing the motor skill in the real world.

However, little work to date has demonstrated the ability of haptic feedback to teach a precise sequence of forces that should be applied as a user moves along a trajectory in space. This type of learning is relevant to force-sensitive, visually-guided tasks, particularly including numerous surgical procedures ([206], [200]). Yokokohji et al [214] presented forces contrary to a correct level of force for an object-manipulation task, but found that this approach was ineffective for the task they were evaluating. More recently, Srimathveeravalli and Thenkurussi [184] used haptic feedback to teach users to replicate both shape and force patterns, but found insignificant benefit of haptic feedback for learning shape patterns, and did not find haptic training to be beneficial at all for learning force patterns.

The present work examines a task in which the participants' goal was to learn and recall a pattern of forces along a single axis. In this context, we demonstrate that

haptic feedback is beneficial for learning a series of forces along a movement trajectory.

## 4.2 Methods

We describe an experiment that assesses the impact of haptic training on participants' ability to learn a sequence of forces. Participants were presented with sequences of forces via three training modalities – visual, haptic, and combined visuohaptic – and were asked to recall those forces. While learning and recalling forces, participants were passively moved along a spatial trajectory, which was also presented visually. The participants used this trajectory as position references for force patterns. A more detailed description of this experiment follows.

### 4.2.1 Participants

Twelve right-handed participants, nine male and three female, aged 19 to 21, took part in the present study. All were undergraduate students. None had previous experience with haptic devices. Participants were compensated with a $5 gift certificate, and an additional $10 gift certificate was offered to the three participants with the highest overall score (across all conditions) as incentive. Written consent was obtained from all participants; the consent form was approved by the Stanford University Institutional Review Board.

### 4.2.2 Apparatus

Visual information was presented on a 19" LCD monitor placed approximately 2.5' from the user. Haptic feedback was presented via an Omega 3-DOF force-feedback device (Force Dimension, Lausanne, Switzerland), resting on a table in front of the monitor. This device was selected because it was able to deliver the sustained forces required for this experiment (up to 8N for up to twenty seconds), which other commercially-available haptic devices could not. Participants were able to rest their elbow on a table. Software was run on a dual-CPU 2GHz Pentium 4 computer running Windows XP, and was developed in C++ using the CHAI toolkit [42]. The

software used for this experiment has been made available online; see Appendix A for download information.

### 4.2.3 Stimuli

The following axis convention was used in the present study:

- The *x* axis runs from the participant's left to the participant's right (parallel to the table)
- The *y* axis runs upward (perpendicular to the table)
- The *z* axis runs toward the user (in and out of the display plane)

Spatial trajectories were generated for each trial to passively move the participant's hand from left to right while sinusoidally varying the participant's hand position along the z axis. The spatial trajectory had no y component; i.e. it was entirely in a plane parallel to the table. Trajectories spanned 10cm in the horizontal (*x*) direction and 6cm in the *z* direction, and moved the user at a constant velocity of 1.6cm/s. The z component of each trajectory was the sum of twenty sinusoids with random frequencies, phases, and DC offsets, with a maximum spatial frequency of 0.3 cycles per centimeter. After summing the sinusoids, each trajectory was scaled to fit the 6cm range in z. A typical spatial trajectory is presented in Figure 22.

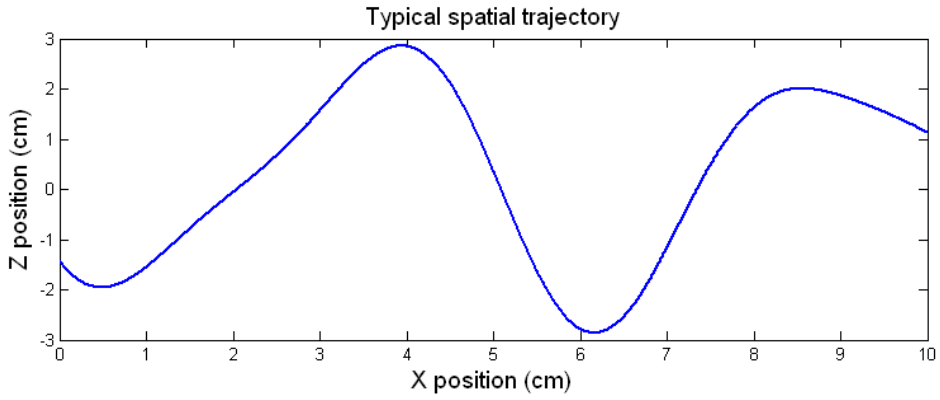



**Figure 22. A typical spatial trajectory used in our experiment.**

Force patterns were generated for each trial along the y axis, perpendicular to the plane of movement along the spatial trajectory. These patterns are the values that the participant was asked to learn in each trial. Force patterns were generated as functions of time, but because the participant was moved along the trajectory at a constant rate, force patterns were also fixed relative to the spatial trajectory. The temporal force patterns were generated as the sum of four sinusoids with random frequencies, phases, and DC offsets, with a maximum frequency of 0.2Hz. After sinusoidal summing, force patterns were scaled into the range [0N,10N]. To introduce limited higher-frequency peaks without creating unreasonably "jagged" force patterns, parabolic "bumps" were randomly blended into each sinusoidal trajectory; these bumps were allowed to range up to 12N. After summing the base pattern and the parabolic bumps, the final force pattern was ramped up and down over the first and last one second of the pattern to avoid jerking the haptic device. A typical force pattern is presented in Figure 23. This graph represents the *magnitude* of the normal force the participant was asked to learn; the learned force was in all cases in the downward (-y) direction.

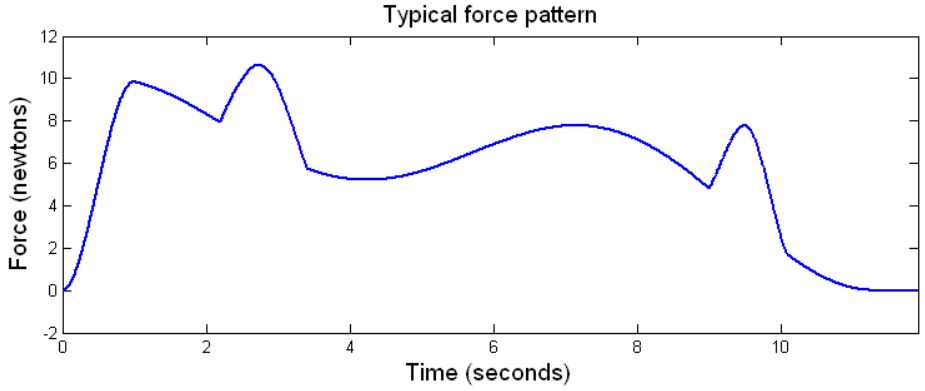


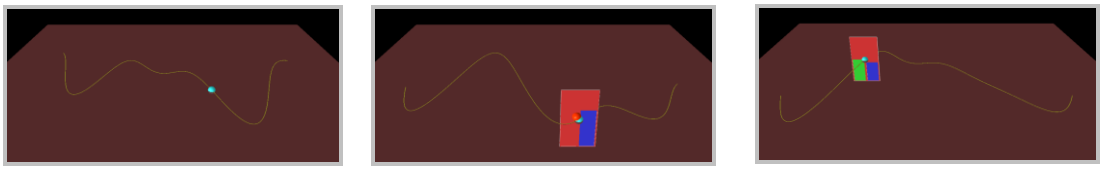

**Figure 23. A typical force pattern used in our experiment.**

### 4.2.4 Experimental Conditions

The following 3 training conditions were employed in a blocked design: haptic display of normal force (H), visual display of normal force (V), and combined visuohaptic display of normal force (V+H). In all three conditions, the participant's

hand was pulled along the spatial trajectory (in the *xz* plane) via a proportional-derivative (PD) controller with proportional and derivative gains of 0.9N/mm and 0.1N/mm, respectively. Offline analysis showed no significant lag behind the ideal trajectory in any participant's data, indicating that the gain was sufficiently high. The visual display showed the spatial trajectory, along with a display of the participant's current device position, under all three training conditions.

In the haptic (H) training condition, the haptic device applied the *opposite* of the embedded force pattern directly to the user along the *y* axis (perpendicular to the movement plane). The participant was instructed to keep the device in the movement plane, i.e. to precisely oppose the upward force applied by the Omega device. In this manner, the participant practiced applying the sequence of forces that he/she was expected to learn. Figure 24a shows the display that was presented to the user in the H condition.



(a) (b) (c)

**Figure 24. The visual representations of the spatial trajectory and normal force presented to the user in the (a) haptic training condition (no representation of force), (b) visual training condition (blue bar representing current target force), and (c) combined visuohaptic training condition (blue bar representing current target force magnitude and green bar current user-applied force magnitude).**

In the visual (V) training condition, the haptic device was constrained to the *xz* plane by a PD controller with P and D gains of 2.0N/mm and 0.3N/mm, respectively. No haptic representation of the embedded force pattern was presented to the user. As the user was pulled along the trajectory, an on-screen blue vertical bar changed its height to indicate the magnitude of the target normal force at the current trajectory position. This bar moved along the trajectory along with the representation of the participant's current device position, so the participant could visually attend to both

simultaneously. Figure 24b shows the display that was presented to the user in the V condition.

In the combined visuohaptic (V+H) training condition, the haptic device was constrained to the xz plane as in the visual (V) condition, and the current target force is displayed as a blue bar, as in the visual condition. However, an additional graphical bar is presented in green. The additional bar indicates the normal force currently being applied by the participant. Participants were instructed to match the heights of the blue and green bars. Thus the participants were – via the plane constraint – receiving haptic feedback equal to the target force pattern. Figure 24c shows the display that was presented to the user in the V+H condition.

A fourth condition – the test (T) condition – was used following all training conditions to evaluate learning through force recall. The visual display in this condition was identical to that used in the haptic (H) condition; no visual indication of force was provided. In the test condition, the haptic device was constrained to the xz plane as in the visual (V) condition. The user was instructed to apply the learned pattern of forces in the y direction (normal to the spatial trajectory).

In all three conditions, a small square appeared on screen when the device reached saturation; this was added to be "fair" to the visual training condition, which otherwise did not provide any indication of absolute force magnitude.

### 4.2.5 Experimental Procedure

Each participant was given an introduction to each of the conditions described above, and was then asked to participate in 72 trials, with a ten-minute break after 36 trials to prevent fatigue. A trial consisted of a single training/testing pair. For each trial, the subject was presented with a trajectory using one of three training conditions (H, V, V+H) and was immediately tested on that trajectory using the test (T) condition described above. Trials were grouped into blocks of three training/testing pairs that repeated the *same* trajectory using the *same* training condition.

For example, for a V condition trial block, the participant was trained with the visual bargraph display of force by traversing the trajectory from left to right once. After returning the stylus tip position to the left of the trajectory, the participant was

immediately tested for force recall once (thus completing one trial). This training/testing pair was then repeated twice more (for a total of three trials per block). A new training condition was then selected and a new trajectory was randomly generated for the next trial block.

In summary, each participant completed a total of 72 trials, representing 24 trial blocks for each of the H, V and V+H conditions.

Throughout the experiment, the device positions and applied normal forces were recorded to disk for offline analysis.

## 4.3 Data Analysis

Each testing trial is scored individually for accuracy of force recall. The input to the scoring mechanism is two force-time curves: the "target" force pattern and the "applied" force pattern. If these curves are similar, the trial should receive a high score for recall accuracy. A simple approach to computing a score might simply subtract the two curves and compute the root-mean-squared (RMS) difference at each point. The synthetic example shown in Figure 25 illustrates why this is an inadequate approach. In this figure, the black line represents a synthetic "correct" force pattern with three clear peaks. The red line represents the force pattern recorded from a hypothetical user who correctly recalled the three force peaks, each with a slight timing error. The green line represents the force pattern recorded from a hypothetical user who didn't apply any force at all. A simple RMS-difference approach to scoring would assign a significantly lower score to the red curve than to the green curve, even though the red curve represents a significantly more accurate recall. Feygin et al [58] computed an optimal linear transformation (scale and shift) to correct for similar errors. This approach, however, will not adequately align all three peaks in this example, because the three peaks are offset in different directions. In other words, different regions of the curve are scaled differently. This problem is even more significant in real data series, which are more complex than this synthetic example.

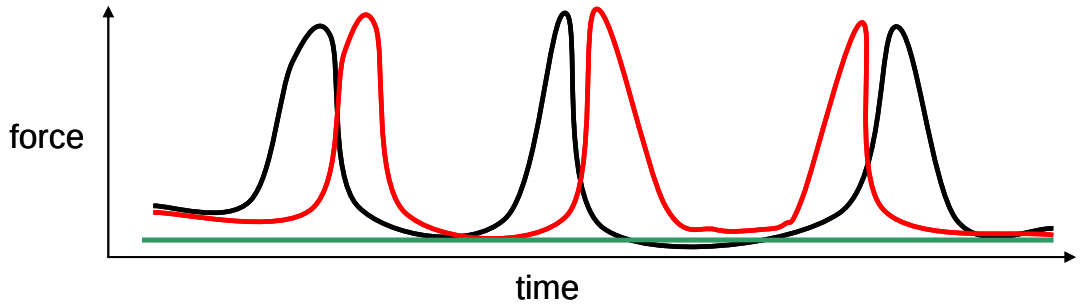



**Figure 25. A synthetic example illustrating the need for non-affine trajectory alignment. The black line represents a synthetic "correct" force pattern. The red line represents the force pattern recorded from a hypothetical user who correctly recalled the three force peaks, and the green line represents the force pattern recorded from a hypothetical user who didn't apply any force at all.**

To address this problem and properly assess recall accuracy participant to local timing errors, we employed a scoring scheme based on dynamic programming (DP). This approach has often been employed to align curves for shape recognition ([13], [138], [154]) and speech recognition [167], and a similar approach was used by Patton and Mussa-Ivaldi [150] for matching "haptic attributes". We describe our adaptation of dynamic programming for aligning force/time curves.

For each trial, the target and applied force patterns are resampled to a common time base, and the applied force patterns are low-pass filtered by a box filter with a width of 100 milliseconds. An error matrix is then constructed to describe how well each point on the target pattern "matches" each point on the applied pattern. If the resampled trajectories are 1000 samples long, this matrix contains 10002 entries. The entry at location (i,j) answers the question: "how similar is point i in the target force pattern to point j in the applied force pattern?" For this experiment, each entry in the error matrix is a weighted sum of the RMS difference in forces and the RMS difference in slopes (df/dt values) between the two points being compared. A penalty value is also specified to the dynamic programming algorithm to penalize time distortions. Dynamic programming is then used to find an optimal (minimum-cost) pairing between samples on the target and applied curves. Figure 26 shows the alignment suggested by dynamic programming for a single trial.

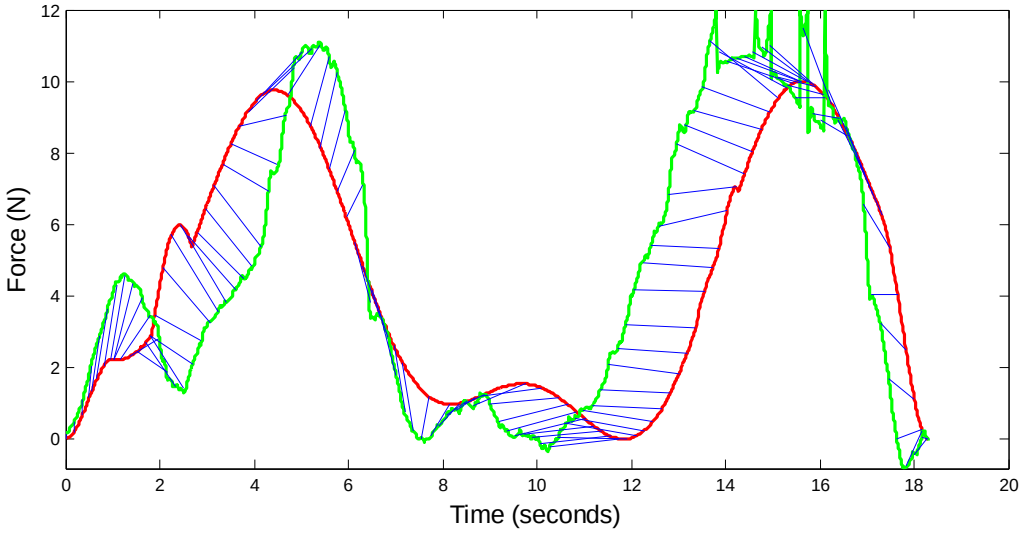



**Figure 26. The alignment computed by dynamic programming for a single trial. The red curve is the target force pattern, the green curve is the applied force pattern, and the blue lines connect points on each curve that are aligned by dynamic programming.**

The applied force pattern is warped according to this alignment to lie on the same time base as the target force pattern. Figure 27 shows the same trial after warping the applied force pattern according to the DP result.

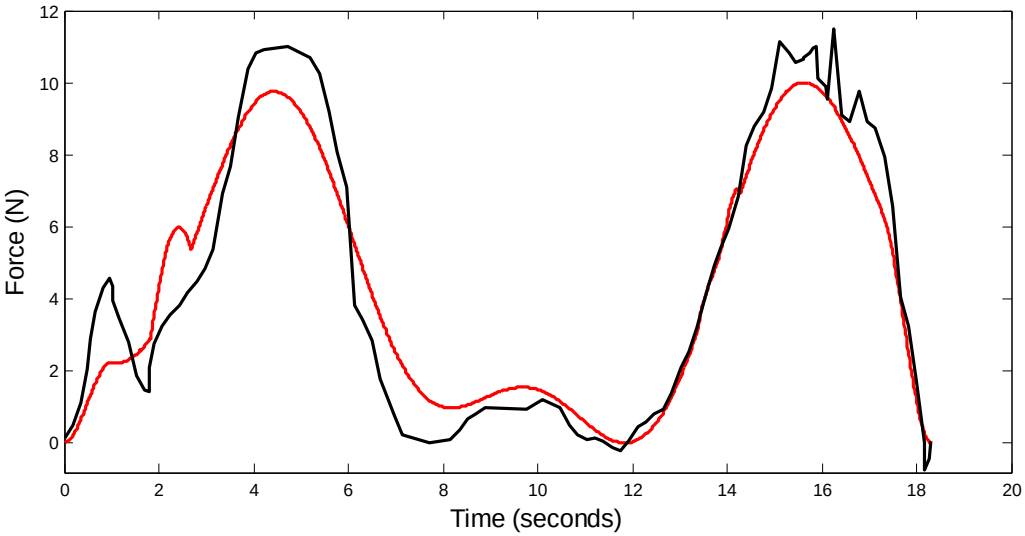


**Figure 27. The target (red) and applied (black) forces for a single trial after warping the applied forces according to the results of dynamic programming (unwarped forces are in Figure 26).**

After DP and warping, a score is assigned to each trial as a weighted average of the DP alignment cost, the RMS difference between the two curves after warping, and the RMS difference between the two curves' slopes after warping. Weights were adjusted empirically to match visual assessments of recall accuracy without knowledge of the experimental conditions for each of the assessed trials. These weighted scores are used to assess the quality of recall for each trial. A score of 0 indicates perfect recall; larger scores indicate lower recall accuracy.

## 4.4 Results

Scores are pooled over each training condition, allowing us to compare the recall quality for each training condition (288 recall trials for each condition). A one-way ANOVA confirms a significant difference among the three training paradigms ($p < 0.001$). Figure 28 shows the mean recall error for each training paradigm with 95% confidence intervals. One-tailed T-tests show that visual training promotes significantly more accurate recall than haptic training ($p=0.002$), and that visuohaptic training promotes significantly better recall than visual training ($p=0.01$).

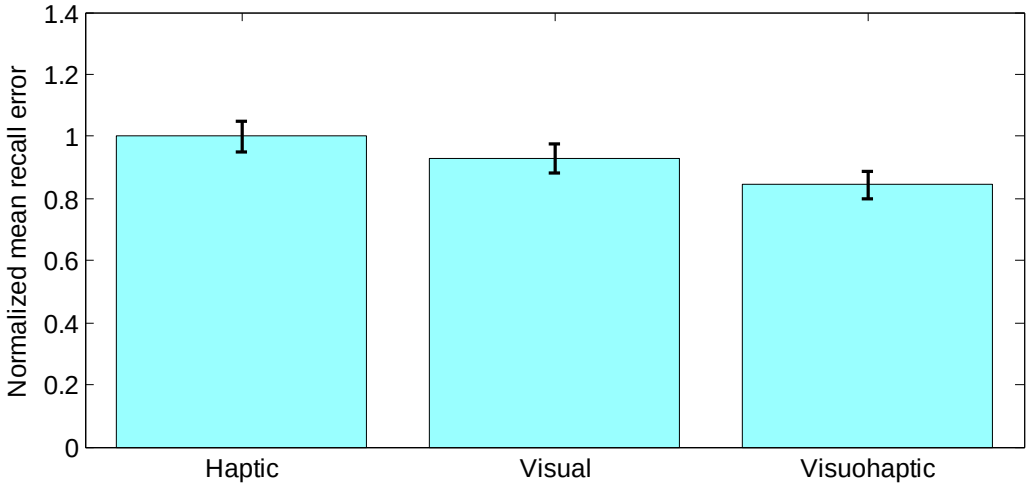


**Figure 28. Mean recall error (in relative units) for each training paradigm. Error bars indicate 95% confidence intervals.**

Table 1 presents the paradigms that promoted the most and least accurate recall for each participant. We observe that 9 of 12 participants had the lowest mean error in the visuohaptic training mode, and only 1 of 12 participants had the highest mean error in the visuohaptic training mode. This is consistent with the results presented in Figure 28, indicating again that visuohaptic training is the most effective paradigm.

| Subject | Best paradigm | Worst paradigm |
|---|---|---|
| 1 | **Visuohaptic** | Visual |
| 2 | **Visuohaptic** | Haptic |
| 3 | Visual | Haptic |
| 4 | **Visuohaptic** | Visual |
| 5 | **Visuohaptic** | Haptic |
| 6 | Visual | **Visuohaptic** |
| 7 | **Visuohaptic** | Haptic |
| 8 | Haptic | Visual |
| 9 | **Visuohaptic** | Haptic |
| 10 | **Visuohaptic** | Haptic |
| 11 | **Visuohaptic** | Haptic |
| 12 | **Visuohaptic** | Haptic |

**Table 1. Training paradigms promoting the most and least accurate mean recall for each subject.**

## 4.5 Discussion and conclusion

The results presented here demonstrate that participants are better able to memorize instructed force patterns when those patterns are presented both visually and haptically, rather than via either modality alone. This is in contrast to the result presented by Srimathveeravalli and Thenkurussi [184], who asked participants to

replicate a force pattern and a position trajectory simultaneously. Their results show that including force information in a skill training paradigm produced *lower* overall error in participants' recall of *positional* information, but *higher* overall error in the participants' recall of *forces*. However, the task they were exploring was significantly more complex: users were asked to recall both force and position in multiple degrees of freedom. Our experiment focused on force alone – with position provided passively as a reference – and only focused on a single axis of force. This more focused task is likely the basis for the difference in results. Additionally, their experiment used smaller movements and a task and device with lower dynamic range, which may have limited participants' ability to recall force information.

Our results also show that haptic training alone is significantly less effective for this task than visual training alone. This is somewhat surprising, since the task is a force-specific task. It is likely that the novelty of memorizing information presented haptically was a confounding factor; visual learning is so pervasive in everyday life that our results may understate the relative potential for learning via haptics alone.

The effectiveness of combined visuohaptic training suggests that haptic training may play an important role in teaching skills like surgery, which are visually-guided but often require different normal and tangential forces to be applied at different places in the workspace. The results presented here suggest a role not only for the use of haptic simulation incorporating simulated environmental feedback, but also active presentation of "correct" forces in a surgical context. These forces may come from online interaction with an experienced instructor, a paradigm we refer to as "haptic mentoring", or from playback of prerecorded forces. Toward this end, we have incorporated the approach presented here into a surgical simulation system [126], and future work will include evaluation of the system's ability to transfer force-sensitive skills to users.

Additionally, we plan to conduct further experiments to explore the roles played by visual and haptic information in the combined visuohaptic training paradigm. This study was designed to evaluate the overall effectiveness of each paradigm in training force patterns, but additional experiments may allow us to

identify that certain frequency components of the target force patterns are being conveyed through one modality or the other.

## 4.6 Software availability

The software used to conduct this experiment runs on Windows XP and is available for download, along with data-manipulation scripts for Matlab and basic documentation, at:

http://cs.stanford.edu/~dmorris/haptic_training

# 5. Automatic Preparation, Calibration, and Simulation of Deformable Objects

Although the procedures simulated by our environment focus primarily on interacting with bone (the topic of Section 3), interacting with deformable tissue constitutes a significant component of any surgical procedure. Modeling deformation at interactive rates has traditionally been one of the most complex problems in computer graphics, and has likely been the most significant technological factor limiting the development and popularization of medical simulation. Many proposed solutions work well for special cases, but do not generalize well or require significant amounts of manual preprocessing.

This section does not claim to solve the complete problem of interactive deformation; rather, it presents a set of solutions that form a complete pipeline from surface mesh to deformable object, focusing on mesh generation, calibration of deformation parameters to a finite element reference model, and interactive rendering. We leverage and extend existing simulation techniques, particularly [199], to address the broader problem of fitting a deformation model into a medical simulation environment. It is our goal to use the approaches presented here to incorporate soft tissue components – such as tumors and deformable components of the inner ear – into the simulator presented in Section 3.

The work presented here was published as [130].

Many simulation environments – particularly those intended for medical simulation – require solid objects to deform at interactive rates, with deformation properties that correspond to real materials. Furthermore, new objects may be created frequently (for example, each time a new patient's data is processed), prohibiting manual intervention in the model preparation process. This paper provides a pipeline for rapid preparation of deformable objects with no manual intervention, specifically focusing on mesh generation (preparing solid meshes from surface models), automated calibration of models to finite element reference analyses (including a novel approach to reducing the complexity of calibrating nonhomogeneous objects), and automated skinning of meshes for interactive simulation.

## 5.1 Introduction and Related Work

### 5.1.1 Background

Interactive physical simulation has become a critical aspect of many virtual environments. Computer games are increasingly using physical simulation to allow players a wider range of interactions with their surroundings; this has become such a prevalent phenomenon that dedicated hardware has become available for rigid body mechanics [156], and software libraries are becoming available to dedicate graphics resources to physical simulation [77]. Simulation for games currently focuses primarily on rigid body dynamics and particle systems (for fluid, smoke, explosions, etc.), but will likely move toward deformable solid simulation as the standard for realism increases.

Many medical simulation environments – both commercial ([182], [36], [87], [183], [191], [213]) and academic [123], [33], [210], [209], [142]) – already depend on modeling deformable solids. The vast majority of tasks performed during surgery involve interaction with deformable bodies, so a medical simulator is expected to not only represent deformation, but to model it with sufficient accuracy for effective training. Force/deformation curves of virtual organs should correspond to their real

counterparts, and deformation should vary realistically among patients, among tissue types, and even within a tissue type.

Currently many of these simulators focus on canonical cases, whose creation requires significant manual intervention by developers, technicians, or manufacturers. As surgical simulation enters mainstream medical practice, the use of patient-specific data in place of canonical cases is likely to become common, allowing a much broader range of applications and training cases. This scenario prohibits the use of tedious manual procedures for data preprocessing. Similarly, as games incorporate more sophisticated simulation techniques, rapid preparation of deformable models will be required to continue the current trend toward player-generated and custom content.

This paper addresses this need: automatic preparation of realistic deformable models for medical simulation and computer games. We restrict our discussion to a particular simulation method in the interest of focusing on *automation* of model preparation (rather than simulation), but the techniques presented here can be generalized to other models.

We assume that the user provides a surface model of the desired structure; this is a reasonable assumption, as surface models are the standard object representation in games and are easily derived from automatically-segmented medical images. We further assume that the user provides constitutive properties describing the material they are attempting to represent; this is also a reasonable assumption, as constitutive properties for a wide variety of materials are available in engineering handbooks. Constitutive properties for biological tissues can be measured experimentally ([32], [168], [194]).

Section 5.2 discusses the generation of volumetric (tetrahedral) meshes from surface meshes. Section 5.3 discusses the use of a finite element reference model to calibrate an interactive simulation. Section 5.4 discusses simulation and rendering, focusing on a geometric interpretation of the simulation technique presented in [199] and a mesh skinning technique that is suitable for our deformation model. The remainder of Section 5.1 discusses work related to each of these three topics.

### 5.1.2 Related Work: Mesh generation

"Mesh generation" generally refers to the process of discretizing a space into volumetric elements. The space is frequently defined by either an implicit or explicit surface boundary, and the elements are generally explicit solid units, commonly tetrahedra or hexahedra when the space is three-dimensional.

Ho-Le [80] provides a summary of core methods in mesh generation for finite element analysis, and Zhang [217] provides a summary of more recent work in this area. Si [181] describes a common, public-domain package for mesh generation, specifically targeted toward finite element analysis applications. Recent work on mesh generation employs physical simulation in the meshing process (e.g. [31]).

The work most closely related to the approach presented in Section 5.3 of this paper is that of Mueller [137], which also focuses on generating approximate, non-conformal meshes for interactive simulation.

### 5.1.3 Related Work: Deformation Calibration

Early work exploring the relationship between non-constitutive simulations (generally mass-spring systems) and finite element analyses began with Deussen et al [51], who optimize a 2D mass-spring system to behave like an analytically-deformed single 2D constitutive element. Similarly, van Gelder [203] analytically derives spring constants from constitutive properties for a 2D mass-spring system. This work also includes a theoretical proof that a mass-spring system cannot exactly represent the deformation properties of a constitutive finite element model.

While most work in this area has been oriented toward volumetric solid deformation using simulation results as a ground truth, Bhat et al [23] use video of moving cloth to calibrate simulation parameters for a cloth simulation. Similarly, Etzmuss et al [55] extend the theoretical approach of van Gelder [203] to derive a mass-spring system from a constitutive model of cloth.

Bianchi et al [24] demonstrate that a calibration procedure can enable a 2D mass-spring system to recover the connectivity of another 2D mass-spring system; deformation constants are held constant. Bianchi et al [25] later demonstrate the

recovery of spring constants, and the 2D calibration of a mass-spring system to a finite element reference model. They do not extend their calibration to 3D, and do not provide a mechanism for handling the exponential growth in optimization complexity associated with 3D objects and complex topologies. Choi et al [38] use a similar approach to calibrate a homogeneous mass-spring system, and Mosegaard [135] uses a similar optimization for simple models but takes dynamic behavior into account during optimization.

### 5.1.4 Related Work: Mesh Skinning

Mesh skinning describes the process of animating the vertices of a rendered mesh to correspond to the behavior of an underlying skeleton. This has become a very common technique for rendering characters in games and video animation; the skeleton often literally represents a character's skeleton and the rendered mesh generally represents the character's skin. Skinning is easily implemented in graphics hardware [144], making it suitable for a variety of simulation environments.

Recent work on mesh skinning has focused on correcting the inaccuracies that result from naïve blending, as per [103], and on automatically associating vertex movements with an implicit underlying skeleton [91] as a form of animation compression. However, bones are generally defined and associated with vertices manually by content developers, as part of the modeling/animation process.

## 5.2 Mesh Generation

This section discusses our approach to generating tetrahedral meshes from surface meshes for interactive deformation.

### 5.2.1 Background

Previous approaches to generating tetrahedral meshes (e.g. [137], [181], [31], [217]) from surface meshes have generally focused on generating conformal meshes (meshes whose bounding surface matches the target surface precisely) for high-precision finite element simulation. Consequently, the resulting meshes are generally highly complex, particularly near complex surface regions.

Interactive simulation presents a different set of requirements and priorities for mesh generation. Since the use of interactive simulation techniques comes with an intrinsic loss in precision, some discrepancy between the target surface mesh and the resulting volumetric mesh is generally acceptable. In particular, the computational expense of increased tetrahedron count does not justify the benefits of a conformal mesh. This is particularly true for applications in games, where physical plausibility and interactivity take precedence over perfect accuracy. For most applications, the surface used for interactive rendering is decoupled from the simulation mesh, so the nonconformality of the mesh will not affect the rendered results (see Section 5.4).

Like finite element simulation, most interactive simulation techniques have difficulties when tetrahedral aspect ratios approach zero. In other words, "sliver" tets are generally undesirable, since they are easily inverted and do not have well-defined axes for volume restoration forces.

The behavior of interactive simulation techniques is often visibly affected by topology, so a homogeneous material is generally most effectively simulated by a mesh with homogeneous topological properties. Thus there is an intrinsic advantage to *regularity* in deformable meshes.

Thus the goal of the technique presented here is to automatically generate nonconformal, regular meshes with large tetrahedral aspect ratios. It is also desirable for the process to proceed at nearly interactive rates for meshes of typical complexity, so the process can easily be repeated following topology changes or plastic deformation during interactive simulation.

### 5.2.2 Mesh Generation

Our mesh generation procedure begins with a surface mesh (Figure 29a), for which we build an axis-aligned bounding box (AABB) hierarchy (Figure 29b).

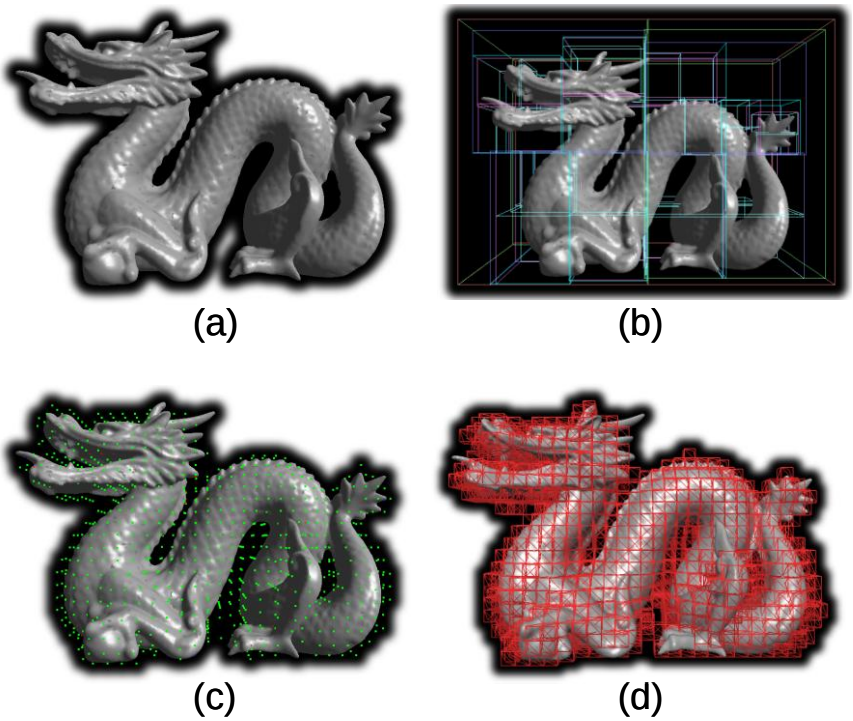


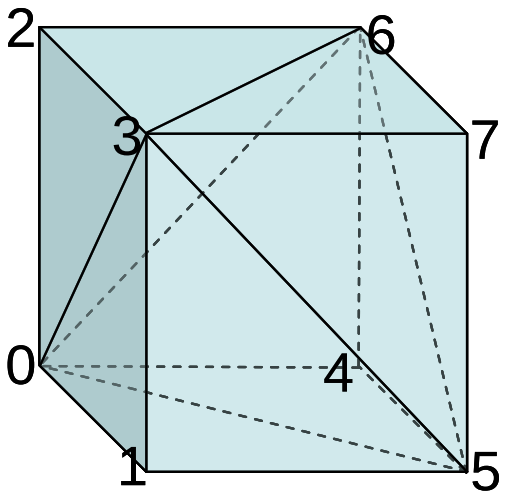

**Figure 29. Stages of the mesh generation process: (a) initial surface mesh, (b) axis-aligned bounding box hierarchy for rapid voxelization (c, with voxel centers in green), and (d) splitting of voxels into tetrahedra.**

The AABB tree is used to rapidly floodfill (voxelize) the surface (Figure 29c). The floodfilling begins with a seed voxel, identified by stepping a short distance along the inward-pointing surface normal of a mesh triangle. This voxel is considered to be an internal voxel. Floodfilling sequentially pulls internal voxels from a queue. A ray is cast from each known internal voxel to each of its neighbors; the AABB hierarchy is used to determine whether this ray crosses the object boundary, with spatial coherence exploited as per [132]. If the ray does not cross the surface, the neighbor is marked as an internal voxel and is placed on the queue. If the ray does cross the surface, the neighbor is marked as a border voxel and is not considered for further processing. Floodfilling proceeds until the queue is empty.

The resolution of voxelization – which determines the resolution of the output tet mesh – is user-specified. Since voxels are isotropic, the user need only specify the voxel resolution of the mesh's longest axis, a simple precision metric that a user can

intuitively relate to the target application. Voxelization is allowed to proceed one voxel outside the surface; for interactive simulation techniques that include collision-detection and penalty-based collision response, it is generally desirable to slightly overestimate object volume at this stage.

Each resulting voxel (defined by its center point) is then used to create a cube of eight vertices. Vertices are stored by position in a hash table; existing vertices can thus be recalled (rather than re-created) when creating a voxel cube, allowing shared vertices in the output mesh. Each resulting cube is then divided into five tetrahedra (Figure 30), yielding the final tetrahedral mesh (Figure 29).




| Tet | Nodes |
|---|---|
| 0 | 0, 1, 3, 5 |
| 1 | 3, 5, 6, 7 |
| 2 | 0, 2, 3, 6 |
| 3 | 0, 4, 5, 6 |
| 4 | 0, 3, 5, 6 |

**Figure 30. Splitting a cube (voxel) into five tetrahedra.**

### 5.2.3 Implementation and Results

The mesh generation approach presented here was incorporated into the *voxelizer* package, available online and discussed in more detail in [132]. The package is written in C++ and uses CHAI [42] for visualization and collision detection (AABB tree construction). Files are output in a format compatible with TetGen [181].

To evaluate the computational cost of our approach, and thus its suitability for real-time re-meshing, we generated tetrahedral meshes for a variety of meshes (Figure 31) at a variety of resolutions on a 1.5GHz Pentium 4. Resolutions were specified as

"long axis resolution", i.e. the number of tetrahedra along the output mesh's longest axis (Section 5.2.2).

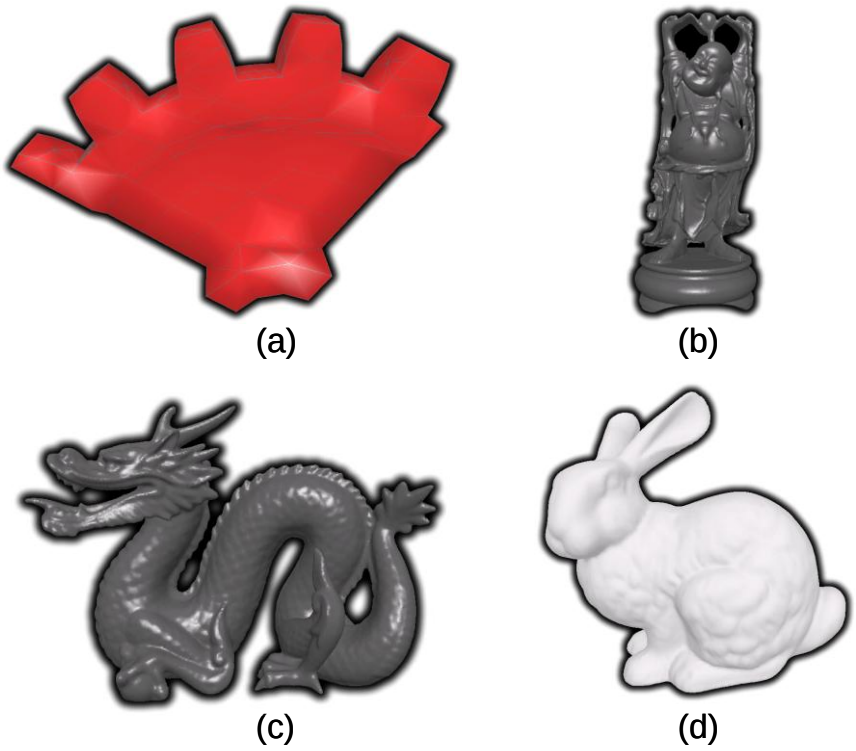



**Figure 31. Meshes used for evaluating mesh generation. (a) Gear: 1000 triangles. (b) Happy: 16000 triangles. (c) Dragon: 203,000 triangles (d) Bunny: 70,000 triangles.**

Table 2 summarizes these results. Mesh generation time is almost precisely linear in output tet count (Figure 33), and mesh generation time is below one second for meshes up to approximately 250,000 tets. Mesh generation proceeds at graphically interactive rates (>10Hz) for meshes up to approximately 20,000 tets. Current parallel simulation techniques ([61], [195]) allow simulation of over 100,000 tets interactively; mesh generation for meshes at this scale is not real-time (about 500ms), but would be sufficiently fast – even at these extremely high resolutions – to allow nearly-interactive background remeshing in cases of topology changes and large deformations. Figure 32 shows mesh generation times as a function of the user-specified precision variable: long axis mesh resolution.

| Input mesh | Input mesh size (triangles) | Long axis resolution (tets) | Output mesh size (tets) | Tetrahedralization time (s) |
|---|---|---|---|---|
| bunny | 70k | 30 | 35840 | 0.153 |
| bunny | 70k | 75 | 478140 | 1.98 |
| bunny | 70k | 135 | 2645120 | 10.139 |
| bunny | 70k | 165 | 4769080 | 18.287 |
| gear | 1k | 30 | 20780 | 0.101 |
| gear | 1k | 75 | 271350 | 1.132 |
| gear | 1k | 135 | 1434065 | 5.789 |
| gear | 1k | 165 | 2504240 | 9.961 |
| happy | 16k | 30 | 10100 | 0.057 |
| happy | 16k | 75 | 126610 | 0.562 |
| happy | 16k | 135 | 662745 | 2.7 |
| happy | 16k | 165 | 1178725 | 4.74 |
| dragon | 203k | 30 | 12750 | 0.083 |
| dragon | 203k | 75 | 158370 | 0.772 |
| dragon | 203k | 135 | 820305 | 3.57 |
| dragon | 203k | 165 | 1453270 | 6.042 |

**Table 2. Tetrahedralization time for the meshes shown in Figure 31, at various output mesh resolutions.**

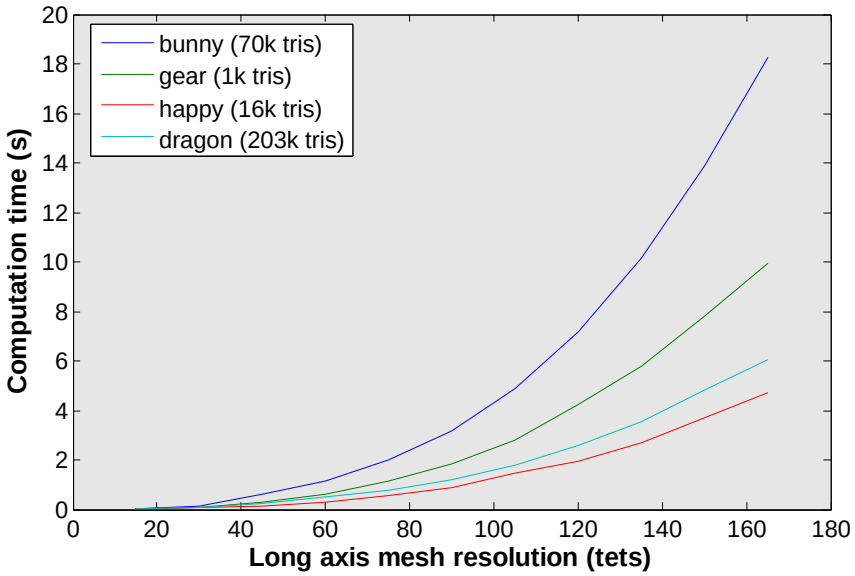


**Figure 32. Mesh generation times at various output mesh resolutions. Long axis resolution, rather than output tet count, is used as the dependent variable; this is an intuitive metric for user-specified mesh precision.**

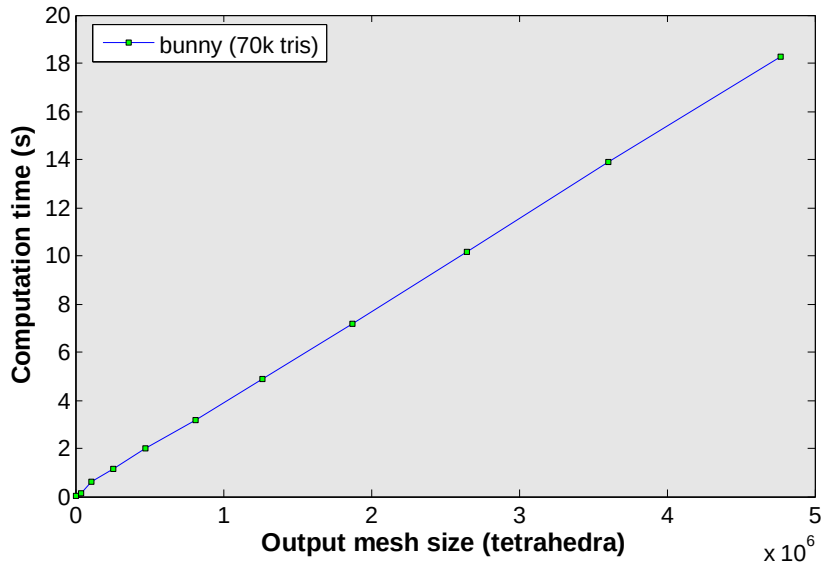



**Figure 33. Mesh generation times at various output mesh resolutions.**

A binary version of our mesh generation approach is publicly available at: http://cs.stanford.edu/~dmorris/voxelizer

## 5.3 Calibration to Ground Truth Deformation

This section discusses the automated calibration of non-constitutive deformation properties using known constitutive properties and a finite-element-based reference deformation.

### 5.3.1 Background

Techniques for simulating deformable materials can be classified coarsely into two categories: constitutive and non-constitutive models.

Approaches based on constitutive models (e.g. [142], [105], [20], [121], [90], [15], [16]) generally use equations from physics to describe how a material will behave in terms of physical constants that describe real materials – e.g. Poisson's

coefficient, Young's modulus, etc. These constants can generally be looked up in an engineering handbook or determined experimentally for a particular material. Many methods in this category are variants on finite element analysis (e.g. [142], [105], [20]), which uses known constitutive relationships between force and deformation to predict how a material will deform. These methods are traditionally very accurate, and are used for computing stresses and strains for critical applications in structural mechanics, civil engineering, automotive engineering, etc. However, these methods are generally associated with significant computational overhead, often requiring solutions to large linear systems, and thus cannot generally be run at interactive rates. When these approaches are adapted to run at interactive rates, they are generally limited in the mesh resolutions they can process in real-time.

In contrast, many approaches to material deformation are non-constitutive, e.g. [199], [133], [134], [61], [195]. Rather than using physical constants (e.g. elastic moduli) to describe a material, such approaches describe objects in terms of constants that are particular to the simulation technique employed. Many approaches in this category are variants on the network of masses and springs, whose behavior is governed by spring constants that can't be directly determined for real materials. In general, these methods are thus not accurate in an absolute sense. However, many approaches in this category can be simulated at interactive rates even for high-resolution data, and these approaches often parallelize extremely well, offering further potential speedup as parallel hardware becomes increasingly common.

In short, the decision to use one approach or the other for a particular application is a tradeoff between realism and performance, and interactive simulations are often constrained to use non-constitutive techniques.

For applications in entertainment or visualization, simulation based on hand-calibrated constants may be adequate. But for high-precision applications, particularly applications in virtual surgery, a deformation model is expected to behave like a specific real material. It is often critical, for example, to teach absolute levels of force that are necessary to achieve certain deformations, and it is often critical to

differentiate among tissue types based on compliance. Thus roughly-calibrated material properties are insufficient for medical applications.

Furthermore, traditional mass-spring systems are usually expressed in terms of stiffnesses for each spring, so the only way to vary the behavior of a material is to vary those stiffnesses. For any significant model, this translates into many more free parameters than a content developer could reasonably calibrate by hand.

Even if sufficient manual labor is available to manually calibrate canonical models, this calibration would generally be object-specific, as much of the deformation properties of a mass-spring network are embedded in the topology and geometry of the network [27]. Therefore calibrated spring constants cannot be directly transferred among objects, even objects that are intended to represent the same material.

The present work aims to run this calibration automatically, using the result of a finite element analysis as a ground truth. While calibration results still cannot be generalized across objects, the calibration runs with no manual intervention and can thus be rapidly repeated for arbitrary sets of objects.

### 5.3.2 Homogeneous Calibration

The following are assumed as inputs for the calibration process:

- A known geometry for the object to be deformed, generated according to the procedure outlined in Section 5.2.
- A known set of loads – defined as constant forces applied at one or more mesh vertices – that are representative of the deformations that will be applied to the object interactively. In practice, these loads are acquired using a haptic simulation environment and an uncalibrated object. Note that a single "load" may refer to multiple forces applied to multiple (potentially disjoint) regions of the mesh.
- Constitutive elastic properties (Poisson's coefficient and Young's modulus) for the material that is to be represented.

The supplied constitutive properties are used to model the application of the specified loads using an implicit finite element analysis, providing a ground truth deformation to which non-constitutive results can be compared. This quasi-static analysis neglects dynamic effects; extension to dynamics is an area for future work.

The same loads are then applied to the same geometry using a non-constitutive simulation, and the simulation is allowed to come to steady-state (a configuration in which elastic forces precisely negate the applied forces). For the implementation presented in Section 5.3.3 we use the deformation model presented in [199], but for this discussion we will treat the simulation technique as a black box with a set of adjustable parameters.

There are, in most cases, large subsets of the parameter space that will not yield stable deformations. In traditional mass-spring systems, for example, inappropriately high constants result in instability and oscillation, while inappropriately low constants result in structural collapse. In either case, local variation in parameters cannot be reliably related to variation in deformation. Optimization will proceed most rapidly if it begins with a baseline deformation that can be used to quickly discard such regions in the parameter space. Therefore, before beginning our optimization, we coarsely sample the parameter space for a fixed number of simulations (generally 100) and begin our optimization with the optimal set among these samples, as per [23] (our optimality metric follows). If none of our samples yield a stable deformation, we randomly sample the space until a stable deformation is obtained.

We then compute an error metric describing the accuracy of this parameter set as the surface distance between the meshes resulting from constitutive and non-constitutive deformation:

$$e_L(\varphi) = \sqrt{\frac{\sum_{i=1}^{nvertices} \left| p_{const}(i) - p_{nonconst}(i) \right|^2}{nvertices}}$$

…where $e_L(\varphi)$ is the error (inaccuracy) for a parameter set φ and load L, *nvertices* is the number of vertices in our mesh, $p_{const}(i)$ is the position of vertex *i* following constitutive deformation, and $p_{nonconst}(i)$ is the position of vertex *i* following non-constitutive deformation with parameter set φ. Note that the non-constitutive deformation is computed once at the beginning of the optimization procedure and is not repeated.

This error metric assumes a one-to-one correspondence between vertices in the two meshes; in practice this is the case for the implementation presented in Section 5.3.3, but were this not the case, the lower-resolution mesh could be resampled at the locations of the higher-resolution mesh's vertices. The deformed positions of the resampled vertices could then be obtained by interpolating the deformed positions of the neighboring vertices in the lower-resolution mesh after deformation (this is analogous to interpolating displacements by free-form deformation [172]).

When multiple loads (to be applied separately) have been defined, we average the resulting errors over those loads to define an accuracy metric for a parameter set:

$$E(\varphi) = \sum_{L=1}^{nloads} e_L(\varphi)$$

…where E(φ) is the average error for the parameter set φ and *nloads* is the number of separate loads to apply. In practice *nloads* is often 1, but we will continue to use the more general E(φ) notation that allows for multiple loads.

The goal of our optimization is thus to find the parameter set φ that minimizes E(φ):

$$\Phi = \arg\min_{\varphi} E(\varphi)$$

…where Φ is our output parameter set, representing the best match to the supplied constitutive parameters for the specified deformations, and φ is bounded by user-specified upper- and lower-bounds, which generally do not vary from problem to problem.

We solve this constrained minimization problem through simulated annealing [100] (SA), a stochastic optimization technique that follows local gradients in a problem space to arrive at minima of the energy function, but periodically jumps against the gradient to avoid local minima. In particular, we use the adaptive simulated annealing [88] (ASA) variant on SA, which automatically adjusts the annealing parameters over time to converge more quickly than traditional SA.

For very simple linear problems, such as identifying the optimal spring constant for a single tetrahedron being stretched in a single direction, we have also employed gradient descent, which is extremely efficient, but complex error landscapes prevent this approach for significant problems. We will discuss further applications for simpler approaches in Section 5.5.

At the completion of the simulated annealing procedure, we will have a non-constitutive parameter set $\Phi$ that optimally matches our non-constitutive deformation to our constitutive deformation. The annealing procedure may take a significant amount of time to complete, but it proceeds with no manual intervention and can thus be efficiently used to prepare numerous deformable models.

### 5.3.3 Implementation

We have implemented the described calibration process using an implicit solver for our constitutive deformation and the method of [199] for our non-constitutive deformation. The finite element package Abaqus [1] is used for reference deformations, and our interactive deformation model is implemented in C++ using CHAI [42] for visualization. Deformation results from both packages are collected in Matlab [111], and optimization is performed with the ASA package [89] through the ASAmin wrapper [166]. Gradients are estimated by finite differencing.

The selected deformation model is described in more detail in Section 5.4; the key point for this discussion is that nodal forces are computed based on four deformation parameters: a volume preservation constant (defined for each tetrahedron), an area preservation constant (defined for each face), a length preservation constant (defined for each edge), and a viscous damping force. These four values are the free parameters for our optimization. For the results presented in

Section 5.3.4, they are taken to be homogeneous throughout the material. Nonhomogeneity will be introduced in Section 5.3.5. In practice, the viscous damping force is always uniform and is allowed to vary only coarsely; once it is calibrated to a reasonable value for a problem, variations should affect the time required to reach steady-state but not the final deformation.

Since we use a quasi-static, implicit simulation for constitutive deformation, we require steady-state results from our non-constitutive simulation as well. A simulation is determined to be at steady-state when the maximum and mean vertex velocities and accelerations are below threshold values for a predetermined amount of time. These values are defined manually but do not vary from problem to problem. Simulations that do not reach steady-state within a specified interval are assigned an error of DBL_MAX.

### 5.3.4 Results: Homogeneous Calibration

We will demonstrate the effectiveness of this approach through a case study, using the problem depicted in Figure 34. Here the base of the gear model is fixed in place (nodes indicated in blue), and opposing forces are applied to the "front" of the gear. This load will tend to "twist" the gear around its vertical axis. The simulated object is defined to be approximately 2 meters wide, with 50 pounds of force applied at each of the two load application points. The constitutive simulation uses a Young's modulus of 100kPa and a Poisson's coefficient of 0.45 .

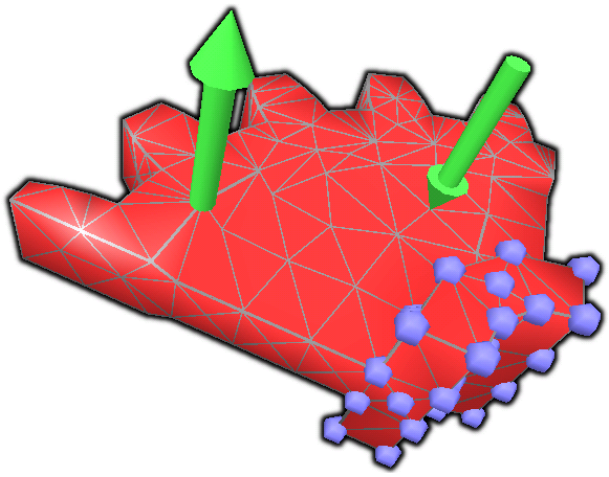


**Figure 34. The deformation problem analyzed in section 3.4. Nodes highlighted in blue are fixed in place; green arrows define the applied load.**

Figure 35 graphically displays the results of the calibration procedure for this problem. The undeformed mesh is shown in Figure 35a. For comparison, the result of a non-constitutive deformation using constants selected through several minutes of manual calibration is presented in Figure 35c. Note that this is not an unreasonable or inconsistent response to the applied loads. Figure 35b and Figure 35d show the results of constitutive deformation and calibrated non-constitutive deformation, respectively. The two models are nearly identical, indicating a successful calibration. Using the error metric described in section 5.3.2, the error was reduced from 0.9 (uncalibrated) to 0.08 (calibrated).

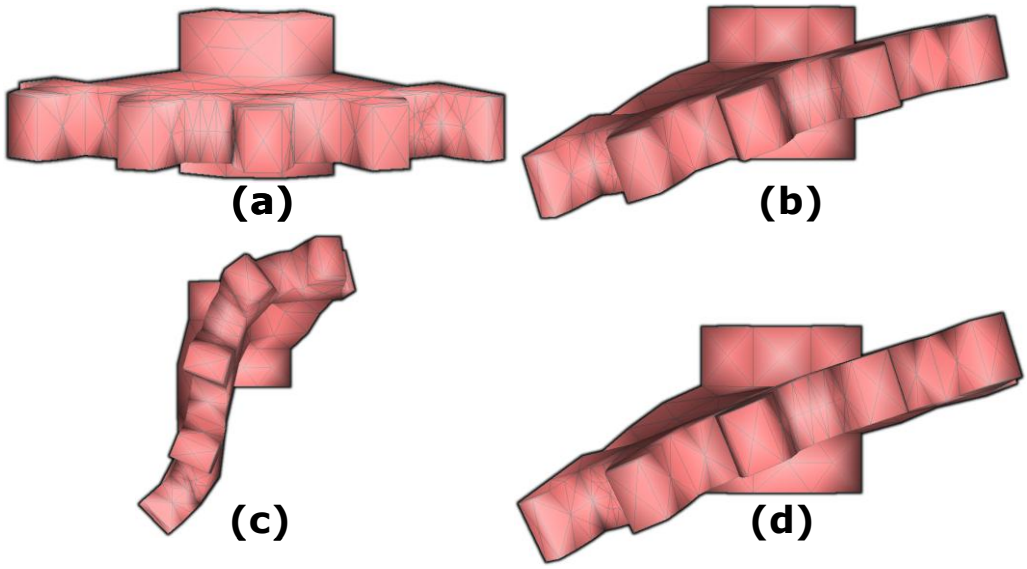



**Figure 35. Results after calibration for the problem shown in Figure 34. Each subfigure shows a "top view" of the model introduced in Figure 34. (a) Undeformed model. (b) Ground truth deformation (resulting from finite element analysis). (c) Baseline non-constitutive deformation (hand-selected constants). (d) Calibrated non-constitutive deformation. (b) and (d) are nearly identical, indicating successful calibration**

Figure 36 looks more closely at the optimization trajectory during this calibration. The optimization proceeds from left to right, with each point representing a simulation pass. Higher values on the y-axis indicate less accurate deformation. The highlighted area indicates the optimization's efficient use of the error gradient for rapid descent from the initial error result. This indicates that a bounded optimization, for which the user specified an acceptable error bound, rather than waiting for a global optimum, would proceed extremely rapidly. This is likely to be the most practical usage model for this approach.

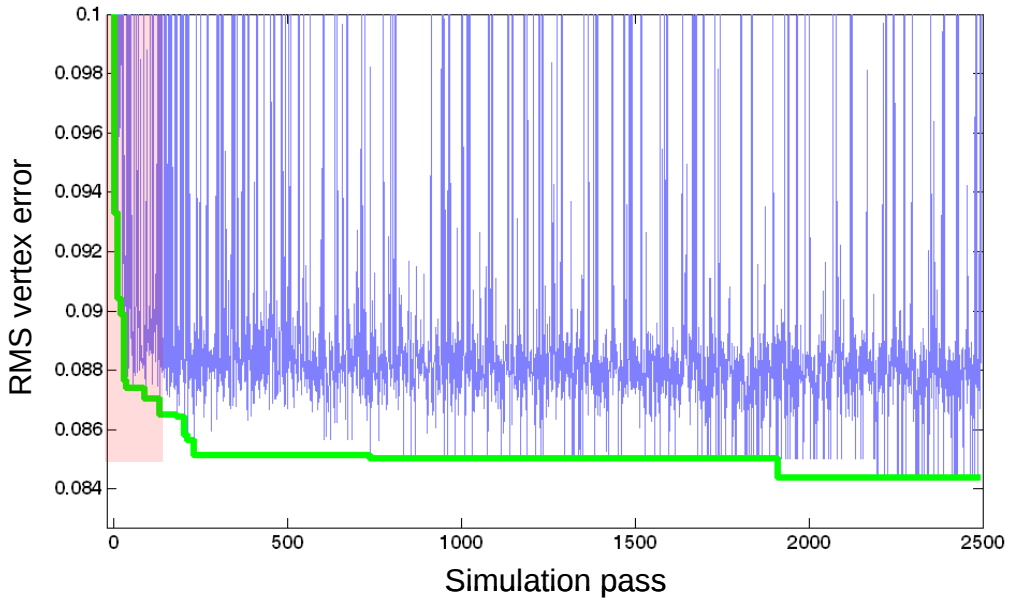



**Figure 36. Optimization trajectory for the calibration shown in Figure 35. Each error value is shown in blue; the green line represents the lower envelope of the blue line, or in other words the best result found so far at any point in the optimization process. The region highlighted in red indicates the rapid initial gradient descent. The y-axis is compressed to improve visibility; the initial error is 0.9, and the maximum error (assigned to non-terminating simulations) is DBL_MAX.**

The "jittery" appearance of the error plot, with numerous simulations resulting in very large errors, results from the annealing process's tendency to occasionally jump from a "good" region of the parameter space to an unexplored region of the

parameter space. These jumps often result in unstable simulations, which are assigned a high error.

Having obtained calibrated constants for this problem, we would like to demonstrate that these constants translate to another load applied to the same object; i.e. we'd like to confirm that our results are not overfit to the particular load on which the system was calibrated.

Figure 37 demonstrates a new load applied to the same model, which will produce an entirely different deformation and will stress the mesh along a different axis. Figure 38 shows the result of transferring the calibration to this problem. Again we present the undeformed mesh and a "baseline" mesh (constants selected quickly by hand) for comparison. We again see an excellent correlation between Figure 38b and Figure 38d, indicating a successful calibration transfer. The RMS vertex error was reduced from 1.0 to 0.1 in this case. The resulting error was thus only slightly higher than the residual self-calibration error represented in Figure 35 and Figure 36.

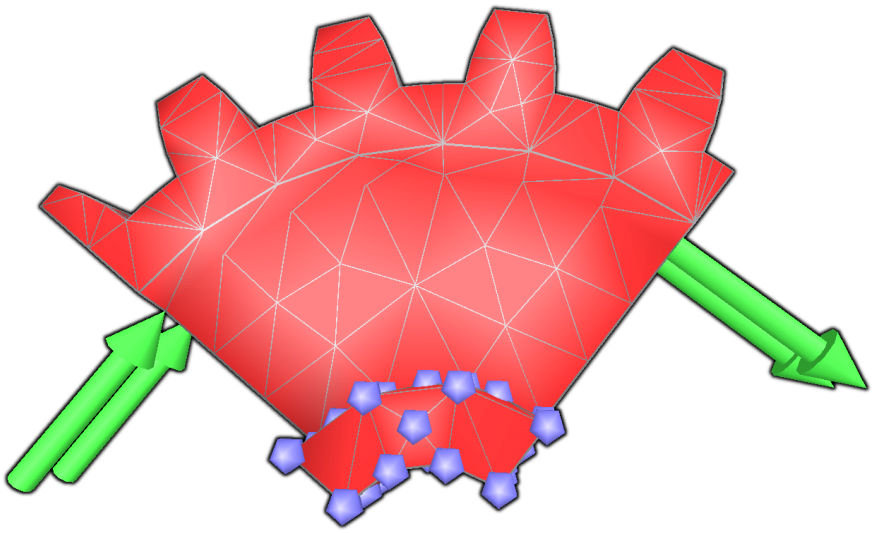


**Figure 37. Calibration verification problem. Nodes highlighted in blue are fixed in place; green arrows define the applied load.**

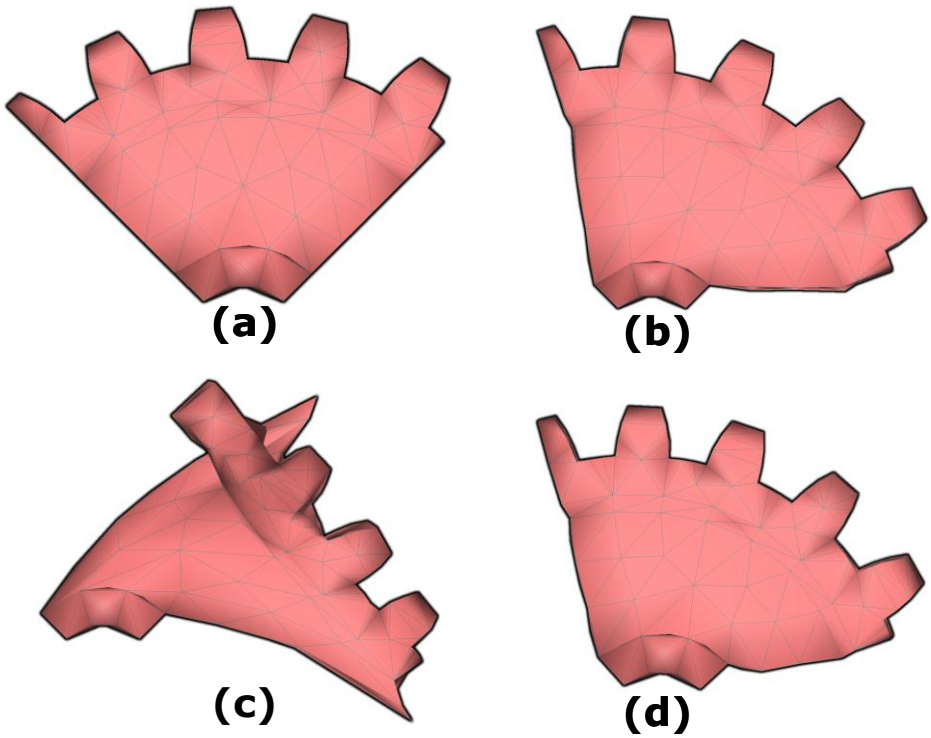



**Figure 38. Calibration verification results. (a) Undeformed model. (b) Ground truth deformation (resulting from finite element analysis). (c) Baseline non-constitutive deformation (hand-selected constants). (d) Calibrated non-constitutive deformation, using the results obtained from the problem presented in Figure 34. (b) and (d) are nearly identical, indicating successful calibration transfer to the new problem.**

### 5.3.5 Nonhomogeneous Calibration

The results presented so far were based on homogeneous materials, i.e. the four calibrated constants were uniform throughout the object. There are, however, two motivations for allowing inhomogeneous deformation constants.

The first is to allow calibration to inhomogeneous reference objects. An object whose material properties vary in space clearly cannot be represented with homogeneous deformation parameters. This is particularly relevant for applications in virtual surgery, where tissues may have material properties that vary according to microanatomy or pathology, or may represent compound materials such as muscle coupled to bone.

A second motivation for allowing inhomogeneous deformation constants is to compensate for deformation properties that are artificially introduced by the geometry and topology of the simulation mesh. van Gelder has shown, for the two-dimensional case, that uniform stiffness properties fail to simulate a uniform object accurately [203]. It is also known that mesh geometry and topology can introduce undesired deformation properties into mass-spring simulations [27]. We would thus like to allow constants to vary within our calibrated mesh, even when it is intended to represent a homogeneous object.

Previous approaches to nonhomogeneous deformation calibration (e.g. [25], [135]) have allowed stiffness constants to vary at each node, which links optimization complexity directly to mesh resolution and presents an enormous optimization landscape.

We present a novel approach to nonhomogeneous parameter optimization, which decouples optimization complexity from simulation complexity and mesh resolution. Specifically, rather than presenting the per-node deformation parameters directly to the optimizer, we allow the optimizer to manipulate deformation parameters defined on a fixed grid; those parameters are then interpolated by trilinear interpolation to each node before every simulation pass. This imposes some continuity constraints on the resulting parameter set (nearby vertices will have similar parameter values), but can greatly speed up the optimization process, making possible the calibration of large meshes that would be prohibitively expensive to optimize per node.

Figure 39 shows an example of the decoupling of the optimization and simulation meshes. The optimization mesh can be arbitrarily simplified to allow, for example, variation of parameters along only one axis of the object (using a $k \times 1 \times 1$ optimization grid).

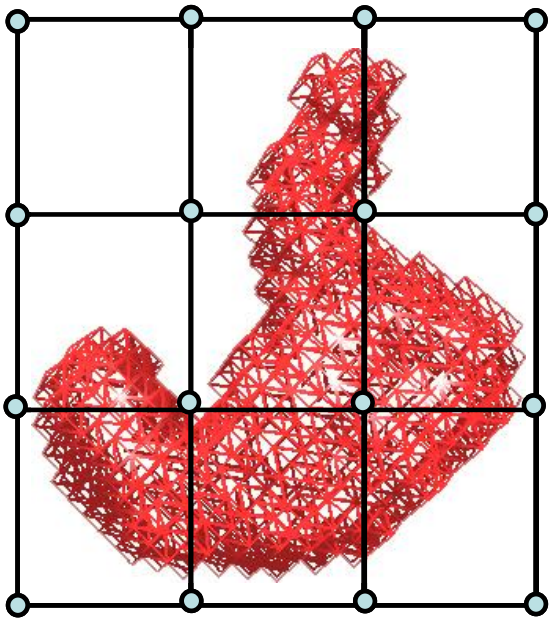


**Figure 39. Decoupled simulation and optimization meshes. The optimizer adjusts constants on the larger grid (blue nodes), which are interpolated to the simulation mesh (red) before each simulation pass.**

As a preprocessing step, each simulation vertex is associated with a set of weights defining the optimization nodes that affect its parameter set. Specifically, we assign weights to the eight optimization nodes that form a cube around each simulation vertex. We will refer to the coordinates of those nodes as $\lfloor x \rfloor, \lceil x \rceil, \lfloor y \rfloor, \lceil y \rceil, \lfloor z \rfloor, \lceil z \rceil$, representing the upper and lower bounds of this vertex's cell in the optimization grid. The coordinates of the eight nodes of this cell are thus:

$$
\begin{array}{ll}
0 & \lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor \\
1 & \lfloor x \rfloor, \lfloor y \rfloor, \lceil z \rceil \\
2 & \lfloor x \rfloor, \lceil y \rceil, \lfloor z \rfloor \\
3 & \lfloor x \rfloor, \lceil y \rceil, \lceil z \rceil \\
4 & \lceil x \rceil, \lfloor y \rfloor, \lfloor z \rfloor \\
5 & \lceil x \rceil, \lfloor y \rfloor, \lceil z \rceil \\
6 & \lceil x \rceil, \lceil y \rceil, \lfloor z \rfloor \\
7 & \lceil x \rceil, \lceil y \rceil, \lceil z \rceil
\end{array}
$$

We then define the cell-relative position of vertex *v* along each axis as:

$$v_{xrel} = (v.x - \lfloor x \rfloor)/(\lceil x \rceil - \lfloor x \rfloor)$$
$$v_{yrel} = (v.y - \lfloor y \rfloor)/(\lceil y \rceil - \lfloor y \rfloor)$$
$$v_{zrel} = (v.z - \lfloor z \rfloor)/(\lceil z \rceil - \lfloor z \rfloor)$$

And the trilinear interpolation weights for this vertex associated with each optimization node are:

| | |
|---|---|
| 0 | $(1 - v_{xrel})(1 - v_{yrel})(1 - v_{zrel})$ |
| 1 | $(1 - v_{xrel})(1 - v_{yrel})(v_{zrel})$ |
| 2 | $(1 - v_{xrel})(v_{yrel})(1 - v_{zrel})$ |
| 3 | $(1 - v_{xrel})(v_{yrel})(v_{zrel})$ |
| 4 | $(v_{xrel})(1 - v_{yrel})(1 - v_{zrel})$ |
| 5 | $(v_{xrel})(1 - v_{yrel})(v_{zrel})$ |
| 6 | $(v_{xrel})(v_{yrel})(1 - v_{zrel})$ |
| 7 | $(v_{xrel})(v_{yrel})(v_{zrel})$ |

Calibration nodes that do not affect parameter values at any vertex (for example, the upper-left calibration node in Figure 39) are discarded and are not used for optimization. In practice, weights are assembled into a (highly sparse) matrix of size [number of calibration_nodes] × [number of vertices] that can be multiplied by a vector of values of length [number of calibration nodes] for each parameter to quickly compute the parameter value at each vertex by matrix-vector multiplication.

Parameter values defined on the optimization grid cannot be used directly for simulation, so to compute a parameter value $p_v$ for a particular simulation vertex v before a simulation pass, we compute the weighted sum:

$$p_v = \sum_{i=0}^{7} w_i p_i$$

…where $w_i$ is the weight associated with node i, as defined above (node numbering here is within a cell, not over the entire grid) and $p_i$ is the parameter value at node i (supplied by the optimizer).

In summary, we learn deformation parameters on a fixed grid, which is generally more sparse than the simulation mesh, and interpolate values to simulation nodes at each evaluation of the error metric. This decouples optimization complexity from mesh complexity.

### 5.3.6 Results: Nonhomogeneous Calibration

We suggested in Section 5.3.5 that nonhomogeneous calibration should improve calibration results even for homogeneous objects. We will thus revisit the problems presented in Section 5.3.4 and assess the benefit of nonhomogeneous calibration.

Figure 40 shows the error reduction for "self-calibration" (the residual error at the completion of optimization) for the two "gear" problems introduced in Section 5.3.5. A significant error reduction is observed in both cases, indicating that the optimizer is able to use the additional degrees of freedom provided through nonhomogeneity. In both cases, a grid resolution of $5 \times 5 \times 3$ was used, where the shortest axis of the gear (the vertical axis in Figure 35a) was assigned to the shortest axis (3 nodes) of the calibration grid.

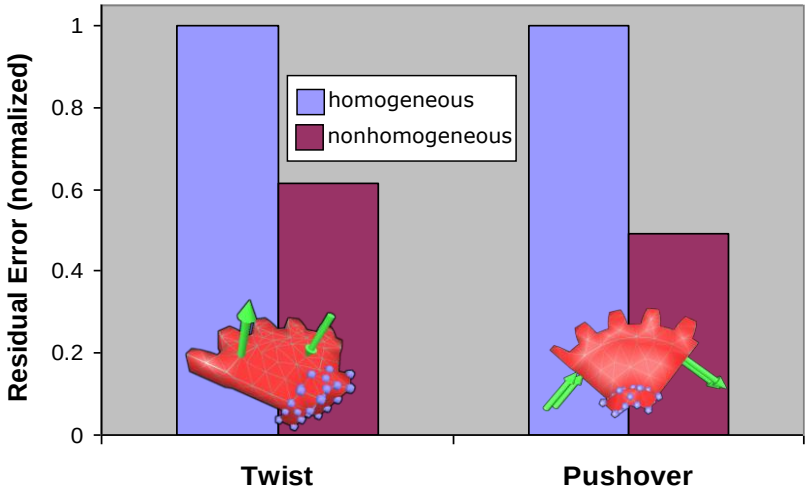



**Figure 40. Improvement in self-calibration due to nonhomgeneous calibration. For each problem, the residual calibration errors following homogeneous and nonhomogeneous calibration are indicated in blue and purple, respectively.**

Having established the benefit of nonhomogeneous calibration for homogeneous objects, we would like to demonstrate the ability of our calibration technique to learn variations in material properties within nonhomogeneous objects.

Figure 41 shows the results of a nonuniform calibration for a cube that was modeled with a uniform Poisson's coefficient (0.499) but included two regions with different Young's moduli (50kPa and 1000kPa) (Figure 41a). An applied load (Figure 41b) resulted in virtually no displacement in the "hard" (bottom) portion of the object according to finite element analysis (Figure 41c). This reference deformation was used to learn constants on an interpolation grid, which converged to the results shown in Figure 41f (values are interpolated to vertices in the figure). Figure 41f shows the distribution of $k_d$; $k_a$ and $k_v$ showed similar distributions, and the damping constant was treated as uniform for this calibration. The resulting non-constitutive deformation (Figure 41e) can be contrasted with the optimal result obtained using *homogeneous* values for all four constants (Figure 41d).

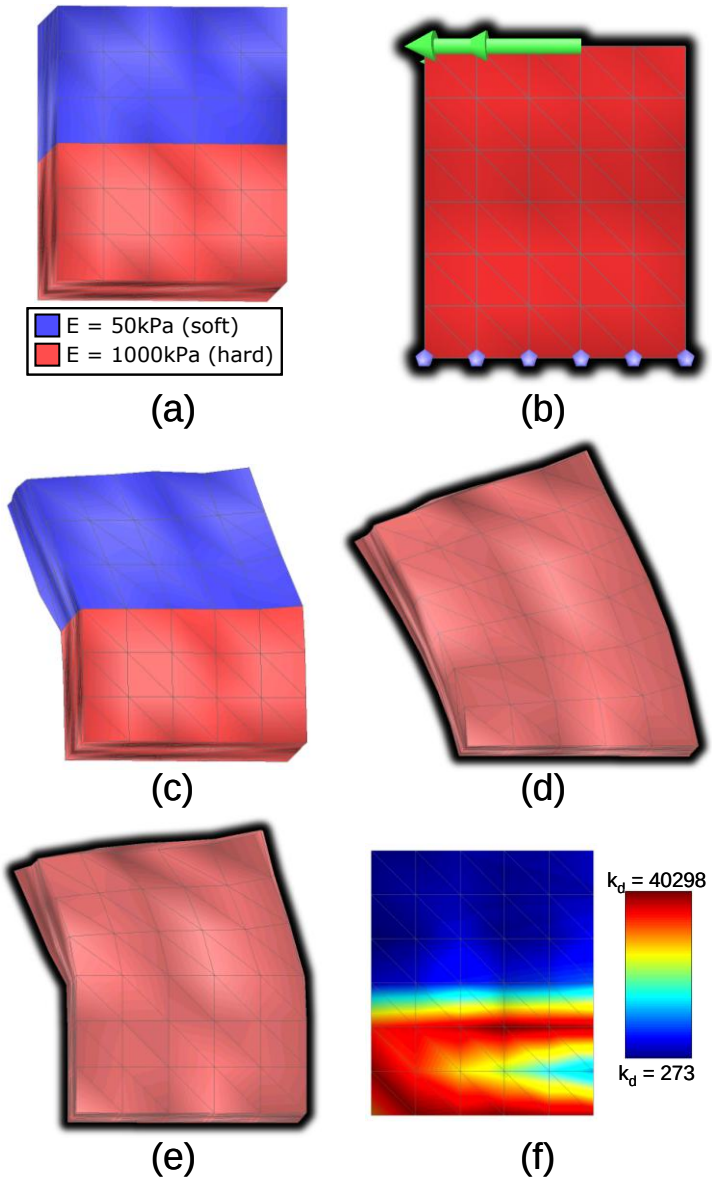



**Figure 41. Results following a nonhomogeneous calibration. The object was modeled with a nonuniform Young's modulus (a), and subjected to the load indicated in (b), with blue highlights indicating zero-displacement constraints. (c) The resulting deformation according to finite element analysis (note that the load is absorbed almost entirely in the "soft" region). (d) The resulting deformation according to a calibrated, non-constitutive model with *homogeneous* parameter values. (e) The resulting deformation according to a calibrated, non-constitutive model with *nonhomogeneous* parameter values; note that the load is correctly absorbed in the top part of the object. (f) The distribution of $k_d$ values after calibration.**

## 5.4 Rendering and Simulation

We have now discussed our approaches to preparing and calibrating tetrahedral meshes and deformation parameters for interactive simulation. This section reviews our selected simulation approach, a reformulation of the method presented in [199]) and discusses our approach to mesh skinning during simulation.

### 5.4.1 Simulation

The deformation model presented in [199] addresses important limitations in traditional mass-spring systems. In particular, local volume-preservation and area-preservation forces, computed for each tetrahedron and each tetrahedral face, respectively, complement traditional length-preservation forces computed along each tetrahedral edge. This model enforces local volume conservation, which approximately constrains global volume, and allows a much wider variety of material behaviors to be expressed than a traditional mass-spring system.

The original presentation of this approach [199] presents these constraint forces as analytic derivatives of energy functions. We will present equivalent geometric interpretations; our implementation is based on these geometric representations of the constraint forces.

#### 5.4.1.1 Distance Preservation

The energy function $E_D$ associated with the distance-preservation force between two connected vertices $v_i$ and $v_j$ is [199]:

$$E_D(v_i, v_j) = \frac{1}{2} k_d \left( \frac{\left| v_j - v_i \right| - D_0}{D_0} \right)^2$$

…where $D_0$ is the rest length of this edge (computed in preprocessing) and $k_d$ is the distance-preservation constant associated with this edge.

The force applied to vertex $v_i$ to minimize this energy is the traditional spring force:

$$F_D(v_i) = k_d\left(\left|v_j - v_i\right| - D_0\right)\left(\frac{v_j - v_i}{\left|v_j - v_i\right|}\right)$$

Intuitively, energy is minimized by shortening or lengthening the spring to its rest length, so we apply a force toward the opposing vertex, whose magnitude depends on the edge's deviation from rest length (Figure 42a).

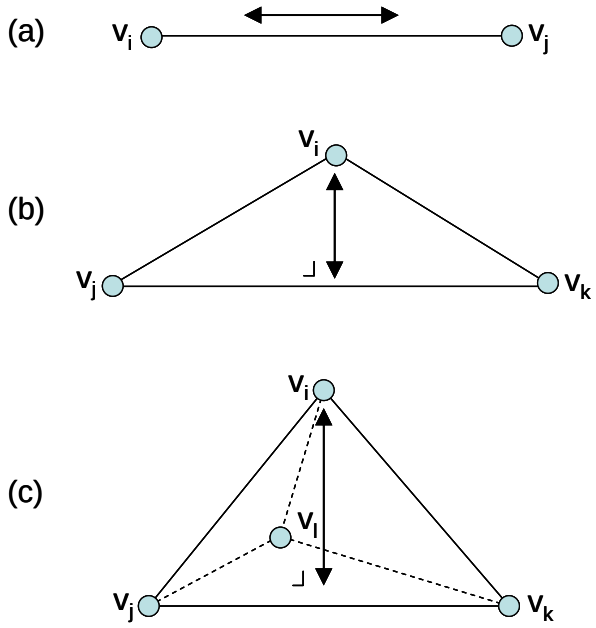



**Figure 42. Geometric representations of energy derivatives with respect to vertex $v_i$, i.e. the direction in which each force should be applied to vertex $v_i$. (a) Distance preservation. (b) Area preservation. (c) Volume preservation. The double-headed arrow indicates force direction in each case.**

In practice, edge lengths are computed before any forces are calculated, so they can be accessed by each vertex without recomputation.

## 5.4.1.2 Area Preservation

The energy function $E_A$ associated with the area-preservation force for the triangle consisting of vertices $v_i$, $v_j$, and $v_k$ is [199]:

$$E_A(v_i, v_j, v_k) = \frac{1}{2} k_a \left( \frac{\frac{1}{2}\left|(v_j - v_i) \times (v_k - v_i)\right| - A_0}{A_0} \right)^2$$

…where $A_0$ is the rest area of this triangle (computed in preprocessing) and $k_a$ is the area-preservation constant associated with this triangle.

To understand the force we should apply to vertex $v_i$ to minimize this energy, we will view this triangle with the edge ($v_j$,$v_k$) on the horizontal axis (Figure 42b). The area of this triangle is equal to ½ times its baseline ( $|v_k - v_j|$ ) times its height. Since the baseline of the triangle cannot be affected by moving vertex $v_i$, the gradient of the triangle area in terms of the position of $v_i$ is clearly along the vertical axis (maximally affecting the height of the triangle). We thus compute this perpendicular explicitly to find the *direction* of the area-preservation force to apply to vertex $v_i$:

$$areagradient(v_i) = (v_i - v_j) - \left( (v_k - v_j) \bullet \frac{(v_k - v_j) \bullet (v_i - v_j)}{(v_k - v_j) \bullet (v_k - v_j)} \right)$$

$$forcedir_A(v_i) = \frac{F_A(v_i)}{|F_A(v_i)|} = \frac{areagradient(v_i)}{|areagradient(v_i)|}$$

Here we have just decomposed the vector ($v_i - v_j$) into components parallel to and perpendicular to ($v_k - v_j$) and discarded the parallel component, then normalized the result (*areagradient*) to get our force direction.

The *magnitude* of this force should be proportional to the difference between the current and rest areas of the triangle. We compute the area as half the cross-product of the edges, i.e.:

$$forcemag_A(v_i) = \frac{1}{2}\left((v_k - v_i) \times (v_j - v_i)\right) - A_0$$

…where $A_0$ is the rest area of this triangle, computed in preprocessing.

And we scale the final force by the area-preservation constant $k_a$ associated with this triangle:

$$F_A(v_i) = k_a \bullet forcemag_A(v_i) \bullet forcedir_A(v_i)$$

In practice, triangle areas are computed before any forces are calculated, so they can be accessed by each vertex without recomputation (area computation also yields triangle normals, which are used for computing volume-preservation forces).

### 5.4.1.3 Volume Preservation

The energy function $E_V$ associated with the volume-preservation force for the tetrahedron consisting of vertices $v_i$, $v_j$, $v_k$, and $v_l$ is [199]:

$$E_V(v_i, v_j, v_k, v_l) = \frac{1}{2} k_v \left( \frac{\frac{1}{6}(v_j - v_i) \bullet ((v_k - v_i) \times (v_l - v_i)) - V_0}{V_0} \right)^2$$

…where $V_0$ is the rest volume of this tetrahedron (computed in preprocessing) and $k_v$ is the volume-preservation constant associated with this tetrahedron.

To understand the force we should apply to vertex $v_i$ to minimize this energy, we will view this tetrahedron with the face $(v_j,v_k,v_l)$ on the horizontal plane (Figure 42c). The volume of this tetrahedron is equal to 1/3 times its base area ( ½ ( $(v_l - v_j) \times (v_k - v_j)$ ) ) times its height. Since the base area of the tetrahedron cannot be affected by moving vertex $v_i$, the gradient of the tetrahedron volume in terms of the position of $v_i$ is clearly along the vertical axis (maximally affecting the *height* of the tetrahedron). We thus compute this perpendicular (the normal to the triangle $(v_j,v_k,v_l)$ ) explicitly to find the *direction* of the volume-preservation force to apply to vertex $v_i$:

$$volumegradient(v_i) = ((v_k - v_j) \times (v_l - v_j))$$

$$forcedir_v(v_i) = \frac{F_v(v_i)}{|F_v(v_i)|} = \frac{volumegradient(v_i)}{|volumegradient(v_i)|}$$

Here we have just computed a vector normal to the triangle $(v_j,v_k,v_l)$ (*volumegradient*) and normalized the result.

The *magnitude* of this force should be proportional to the difference between the current and rest volumes of the tetrahedron. We compute the volume of the tetrahedron and subtract the rest volume:

$$forcemag_V(v_i) = \frac{1}{6}\left(\left(v_j - v_i\right) \bullet \left((v_k - v_i) \times (v_l - v_i)\right)\right) - V_0$$

…where $V_0$ is the rest area of this triangle, computed in preprocessing.

And we scale the final force by the volume-preservation constant $k_v$ associated with this tetrahedron:

$$F_V(v_i) = k_v \bullet forcemag_v(v_i) \bullet forcedir_v(v_i)$$

In practice, tetrahedral volumes are computed before any forces are calculated, so they can be accessed by each vertex without recomputation.

As is described in [199], these forces are accumulated for each vertex and integrated explicitly using Verlet integration. A viscous damping force is also applied to each vertex according to a fourth material constant $k_{damp}$.

### 5.4.2 Mesh Skinning

The tetrahedral mesh used for simulation will generally present a lower-resolution surface than the original mesh; rendering this surface directly significantly limits rendering quality (compare Figure 43a to Figure 43b). It is thus desirable to decouple the rendering and simulation meshes by "skinning" a rendering mesh onto a simulation mesh (Figure 43c).

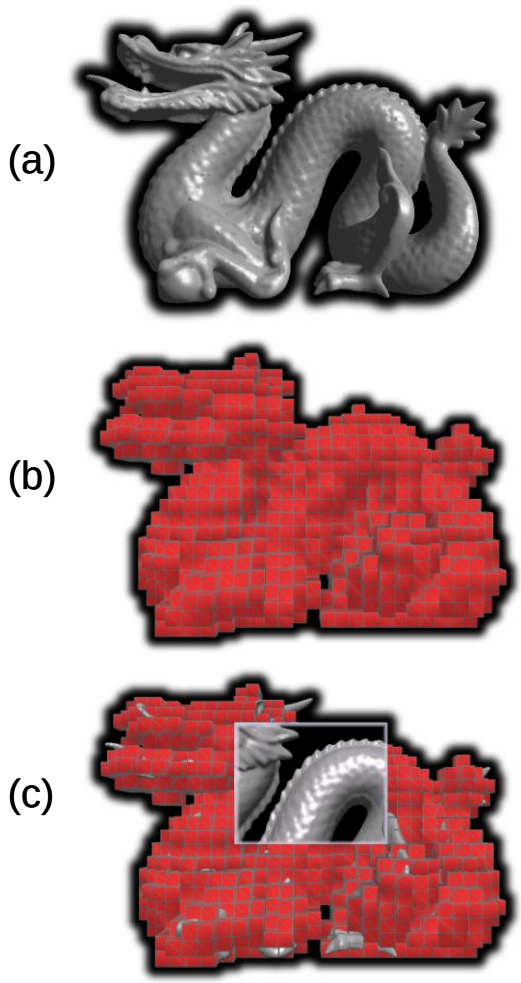



**Figure 43. Skinning a rendering mesh on a simulation mesh. (a) Original mesh, used for interactive rendering. (b) Tetrahedral mesh, used for interactive simulation. (c) Rendering mesh skinned on simulation mesh (with cutaway view).**

This type of mesh skinning is common for applications that have a low-resolution rigid skeleton for animation and wish to deform a rendering mesh to reflect the movements of the underlying bones, an operation that can be performed on commodity graphics hardware [144]. However, such approaches assume a low-degree-of-freedom underlying skeleton and are thus not suitable for skinning complex meshes. Furthermore, mesh skinning usually involves manual assignment of vertices to one or more bones, which is not practical when the set of independently deforming components is very large. In other words, manually assigning vertices to be controlled by specific tetrahedra would be prohibitively time-consuming.

We thus present an automatic mechanism for skinning a rendering mesh onto a simulation mesh. Our approach is similar to free-form deformation [172], which determines the movement of vertices in a deforming space defined by a grid of control points. In our case, the physically-based deformation of the tetrahedral mesh defines a deforming space, and the vertices of the rendering mesh are translated accordingly.

Specifically, we perform a preprocessing step that begins with defining a "vertex-space" coordinate frame for each vertex $v_s$ on the surface of the simulation mesh. We assume that surface vertices in the simulation mesh are tagged as such during the mesh generation process (Section 5.2). The vertex-space coordinate frame $\mathbf{F_{vs}}$, with origin at $v_s$, is defined by the three reference vectors $\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$, which are created as follows and are orthogonal by construct (Figure 44):

- $\mathbf{F_x}$: the surface normal at $v_s$. Surface normals are computed before and during simulation by averaging the face normals of all triangles that contain $v_s$.
- $\mathbf{F_y}$: The component of first "surface edge" connected to $v_s$ that is perpendicular to the normal at $v_s$. A "surface edge" is defined as an edge that connects to another vertex that is on the surface of the mesh. This component is computed as follows:

$$F_y = \left(v_{opposite} - v_s\right) - \left(\left(\left(v_{opposite} - v_s\right) \bullet N\right)N\right)$$

  …where $v_s$ is the vertex at which we're defining a frame, $v_{opposite}$ is the simulation vertex at the other side of the selected surface edge, and **N** is the unit normal vector at $V_s$. $\mathbf{F_y}$ approximates a local surface tangent vector.
- $\mathbf{F_z}$: The cross-product of $\mathbf{F_x}$ and $\mathbf{F_y}$.

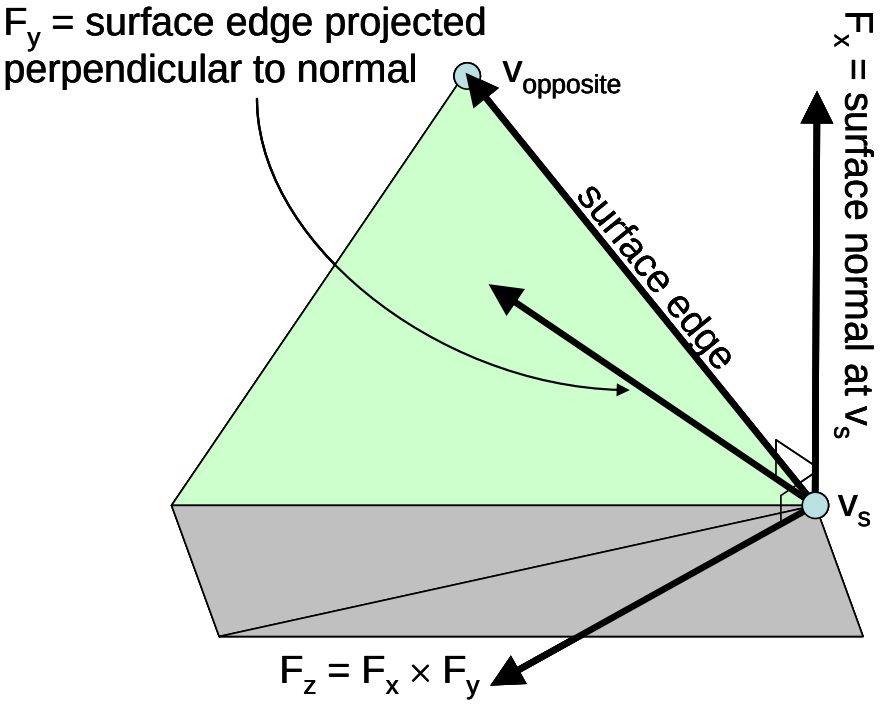



**Figure 44. Vertex-space coordinate frame definition. The triangles shown in gray, and their edges, are not used explicitly for defining this vertex's coordinate frame, but will influence the frame through their influence on the surface normal.**

$\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$ are each normalized to yield an orthonormal basis. Note that in practice, coordinate frames are not defined until vertices are used in subsequent steps, so that coordinate frames are not computed for vertices that are not used for skinning. We have presented coordinate-frame definition first for clarity.

After defining coordinate frames, we place all simulation vertices on the surface of the simulation mesh in a kd-tree [19].

For each vertex on the rendering mesh, we then find the nearest *nneighbors* vertices on the surface of the simulation mesh. Higher values for *nneighbors* result in more expensive rendering but more accurate rendering mesh deformation. In practice we generally set *nneighbors* = 5.

For each vertex $v_r$ on the rendering mesh, and each of its nearby vertices $v_s$ on the simulation mesh, we then compute the world-frame offset of $v_r$ relative to $v_s$, and rotate it into the coordinate frame $\mathbf{F_{vs}}$ defined at $v_s$:

$$offset_{world}(v_r, v_s) = v_r - v_s$$

$$R_{v_s} = \begin{bmatrix} F_x.x & F_x.y & F_x.z \\ F_y.x & F_y.y & F_y.z \\ F_y.x & F_z.y & F_z.z \end{bmatrix}$$

$$offset_{vertex}(v_r, v_s) = \left[R_{v_s}\right] offset_{world}$$

…where $\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$ are the components of $\mathbf{F_{vs}}$, as computed above. We store $\mathbf{offset}_{vertex}(v_r,v_s)$ for each of the *nneighbors* $v_s$'s associated with $v_r$. We also compute, for each $\mathbf{offset}_{vertex}(v_s,v_r)$, a weighting factor defined by the distance between $v_s$ and $v_r$ (closer vertices should have more influence over $v_r$). The weighting factor for a particular $(v_r,v_s)$ is computed as:

$$w(v_r, v_s) = \frac{\dfrac{1}{\left|v_r - v_s\right|^2}}{\sum_{i=1}^{nneighbors} \dfrac{1}{\left|v_{r_i} - v_s\right|^2}}$$

…where the denominator here is a normalization factor, ensuring that the *nneighbors* weights add up to 1.

The indices of all weighted vertices, the weight values, and the $\text{offset}_{vertex}$ values are stored for each rendering vertex $v_r$.

During each frame of interactive rendering, for each vertex $v_r$, we look up the indices and deformed positions of each weighted vertex $v_s$. Then to find the position at which $v_r$ should be rendered, we recompute each coordinate frame $\mathbf{F_{vs}}$ exactly as described above (including normalization) using the *deformed* position of $v_s$, yielding new $F_x$, $F_y$, and $F_z$ vectors (which we'll refer to as $F_x$', $F_y$', and $F_z$'). The new position for $v_r$ based on a simulation vertex $v_s$ is then computed as:

$$p(v_r, v_s) = F_x' \bullet offset_{vertex}.x + F_y' \bullet offset_{vertex}.y + F_z' \bullet offset_{vertex}.z$$

The coordinate frame is based on the local surface normal and the local tangent vector (the chosen surface edge), and thus rotates with the space surrounding $v_s$.

The final position for $v_r$ is the weighted average of the position "votes" from each $v_s$:

$$p(v_r) = \sum_{i=1}^{nneighbors} w(v_r, v_{s_i}) \bullet p(v_r, v_{s_i})$$

### 5.4.3 Implementation and Results

The proposed simulation approach is a reformulation of [199], so we refer to their results for detailed deformation results. The proposed skinning approach was implemented using CHAI [42] for visualization and the ANN library [136] for kd-tree searching. With N=5 and a the simulation and rendering meshes shown in Figure 15 (50,000 rendered faces and 13,000 simulated tetrahedra), simulation proceeds at 200fps, with rendering taking place every 10 simulation frames (20fps).

Skinning results are best communicated by video, so we have made a video of our skinning approach, applied during interactive deformation, available at:

http://cs.stanford.edu/~dmorris/video/dragon_deforming.avi

## 5.5 Conclusion and Future Work

We have presented an automated pipeline for interactively deforming an object originally defined as a surface mesh. Pipeline stages included mesh generation, calibration to a constitutive model using simulated annealing, and simulation/rendering.

### 5.5.1 Future Work: Mesh Generation

Future work on mesh generation will focus on generating nonuniform meshes that provide more resolution in more detailed regions of the surface model; the AABB hierarchy that we already create during voxelization provides a multiresolution

representation of the object that translates naturally into a voxel array. Calibration (Section 5.3) will compensate for simulation artifacts resulting from nonuniform mesh resolution. Also, simulations that involve topology changes (cuts and fractures) and large deformations may benefit from dynamic background re-meshing, another area for future research.

### 5.5.2 Future Work: Calibration

Our calibration procedure is currently naïve to the deformation model and treats each error function evaluation as a black box. Calibration would be greatly sped up by automatically and dynamically generating loads that probe sensitive, high-resolution, or user-highlighted regions of the mesh. Also, error gradients are currently estimated by finite differencing; more sophisticated approaches would adjust constants more efficiently using ad hoc heuristics that predict the effects of parameter changes (for example, higher stiffness constants are likely to reduce overall deformation).

Additionally, a more sophisticated error metric would penalize shape deformation but allow rigid body transformations; the current per-vertex-distance metric penalizes all errors equally. The calibration could also derive a more accurate seed point for optimization by using simple, canonical models (for example, homogeneous cubes or single tetrahedra) to obtain approximate canonical values for deformation constants representing particular material properties.

Non-geometric error metrics that incorporate stress or surface tension would also improve the applicability of our approach to applications that require force information, e.g. simulations incorporating haptics or fracture/cut modeling.

Another application of the presented approach is topology optimization. The ability to find optimal constants for a given topology can be generalized to iteratively adjust topology, to minimize mesh size and simulation complexity while still satisfying a given error bound.

We would also like to generalize our calibration approach to more complex deformation models, particularly incorporating dynamics, nonlinear stress/strain relationships, plasticity, and topology changes.

### 5.5.3 Future Work: Parallelization

Finally, all of the approaches presented in this paper lend themselves extremely well to parallelization, and are expected to benefit from parallel implementations. Voxelization can be parallelized across regions at a high level, or across AABB nodes at a finer level. A custom annealing procedure could make use of multiple, simultaneous samples in the parameter space, and would be further optimized by a parallelized version of the simulation itself, as per [61]. The skinning approach presented in Section 5.4 is particularly well-suited to parallel implementation on graphics hardware, especially when using a simulation technique such as [61], [195], [133], or [134], which place vertex positions in a GPU-resident render target that can be accessed from a vertex shader used to transform the vertices of the rendering mesh.

### 5.5.4 Supplemental Material

The mesh generation approach presented in Section 5.2 is available in binary form at:
http://cs.stanford.edu/~dmorris/voxelizer

A video of our mesh skinning approach is available at:
http://cs.stanford.edu/~dmorris/video/dragon_deforming.avi

The “dragon”, “bunny”, and “happy” models were obtained from the Stanford 3D Scanning Repository [82]. The “gear” model was obtained from the TetGen examples page [83].

# 6. Algorithms and Data Structures for Haptic Rendering: Curve Constraints, Distance Maps, and Data Logging

This section presents three algorithms that were developed in the course of building the architecture used for Sections 3, 4, and 5. Haptic curve constraints (Section 6.2) are a natural extension of the haptic guidance techniques presented and evaluated in Section 4; future experiments will involve repeating the work presented in Section 4 under this sort of constraint. Distance fields were experimented with as a "fail-safe" (to handle excessive penetration depths) for the haptic force computation techniques presented in Section 3, and the distance-field generation mechanism presented in Section 6.3 fits into the voxelization architecture presented in Section 5. The data logging technique described in Section 6.4 was critical to the experiments presented in Section 3 and Section 4, both of which required high-bandwidth data logging on a high-priority haptic thread.

The algorithmic subsections in this section are presented largely as pseudocode, intended to allow a reader to quickly implement the described techniques. A C++ implementation of our data-logging system is also available online and is linked from the end of Section 6.4; binary versions of the approaches presented in Sections 6.2 and 6.3 are available online and are linked from their respective sections.

The work presented here was published as [132].

In this section, we describe three novel data processing techniques used for haptic rendering and simulation:

- We present an approach to constraining a haptic device to travel along a discretely-sampled curve.
- We present an approach to generating distance maps from surface meshes using axis-aligned bounding box (AABB) trees. Our method exploits spatial coherence among neighboring points.
- We present a data structure that allows thread-safe, lock-free streaming of data from a high-priority haptic rendering thread to a lower-priority data-logging thread.

We provide performance metrics and example applications for each of these techniques. C++-style pseudocode is provided wherever possible and is used as the basis for presenting our approaches. Links to actual implementations are also provided for each section.

## 6.1 Introduction

Applications incorporating haptic feedback are subject to significant performance constraints; it is generally accepted that an application needs to sustain a 1kHz haptic update rate before sampling effects become perceptible.

This stringent computation-time limitation requires careful consideration of the design and implementation of preprocessing, rendering, and data streaming techniques. In this section, we present three techniques for optimized haptic data processing, each in an individual subsection. Section 6.2 will discuss the implementation of a haptic curve constraint, or "virtual fixture", using kd-trees. Section 6.3 will discuss the rapid (offline) generation of exact signed distance fields for surface meshes. Section 6.4 will discuss a threaded data structure for lock-free

streaming of data from a high-priority haptic rendering thread to a lower-priority disk-interaction thread.

## 6.2 Haptic Curve Constraints

### 6.2.1 Background

Haptic devices generally provide a user with three or six degrees of freedom. Haptic feedback, however, offers the possibility of dynamically reducing the effective degrees of freedom available within the device's workspace via virtual constraints.

Non-penetration constraints associated with surfaces are extremely common and are used in nearly every haptic simulation involving interaction with rigid objects, but other types of constraints have been applied using haptic devices as well. Abbott et al [2] propose "virtual fixtures" to assist in dexterous manipulation; the goal is to reduce the degrees of freedom involved in a complex task and/or to restrict a device's motion to a "safe" portion of the workspace. This may be particularly suitable for robotic surgery applications in which an actuated master can assist the surgeon by restricting the movement of the slave. The authors discuss multiple types of fixtures, including a "guidance virtual fixture" (GVF), which is a constraint associated with a 3D curve. Garroway and Hayward [60] constrain the user to an analytic curve to assist in editing a spatial trajectory.

In both of these cases, it is assumed that the closest point on the curve to the current haptic probe position and/or the distance to that point are readily available, either by analytic computation or by explicitly tracking the progress of the haptic probe along the curve.

### 6.2.2 Discretized Curve Constraints

For some applications, particularly those where constraints can be dynamically added and removed, it may be necessary to constrain a user to a curve beginning at an arbitrary starting point, or to recover when the constraint has been significantly violated. It is thus necessary to rapidly find the closest point on a curve to the current haptic probe position.

In addition, analytic representations are not always available for curves; curves are most generally represented as discretely-sampled, ordered point sets rather than analytic functions. This is particularly useful, for example, for haptic training applications (e.g. [211], [58], [129]), in which one might use previously-recorded trajectories as teaching examples.

We thus provide a rapid method for finding the closest point on a discretely-sampled curve to a current probe position. We also present an approach to tracking the constraint position on a curve when the haptic device may deviate from the constraint and approach other points on the curve to which it should not become constrained. Tying the haptic device to this constraint position by a virtual spring will provide a general-purpose curve constraint.

A curve is assumed to be represented as a series of N points, each of which stores its 3-dimensional position, an index into a linked-list or flat array that stores the N points in order, and its arcposition along the curve (the curve runs from arcposition 0.0 to arcposition 1.0). The curve is not required to have a uniform sampling density. Each point $p_i$ (for $i \neq 0$ and $i \neq (N-1)$ ) is implicitly part of two line segments, $[p_{i-1} \rightarrow p_i]$ and $[p_i \rightarrow p_{i+1}]$. For clarity, I will provide C++ pseudocode of the relevant data structures and computations throughout this section, beginning with the representation of the curve and sample points. The 'vector' class is assumed to represent a 3-dimensional vector and to support the expected operators.

```
struct curvePoint {
  vector pos;
  unsigned int index;
  float arcpos;
};

struct curve {
  unsigned int N;
  curvePoint* points;
};
```

All of these points are then placed in a standard kd-tree [19] (a 3d-tree in this case). A kd-tree stores a point set and provides efficient retrieval of the subset of points that lie

within a bounding rectangle. This can be generalized at minimal cost to return the approximate nearest K neighbors to a given test point. We will assume that our kd-tree provides the following function, which returns the *K* points closest to *testPoint*:

```
void search(vector testPoint,int K, curvePoint* points);
```

At each timestep at which a haptic constraint force is requested, we use this interface to find the N closest points to the device position $p_{dev}$. N is chosen empirically; higher values of N require more computation time but reduce the occurrence of incorrect forces resulting from sparse sampling of the curve. Figure 45 demonstrates this problem and illustrates why using N=1 does not generally give correct results.

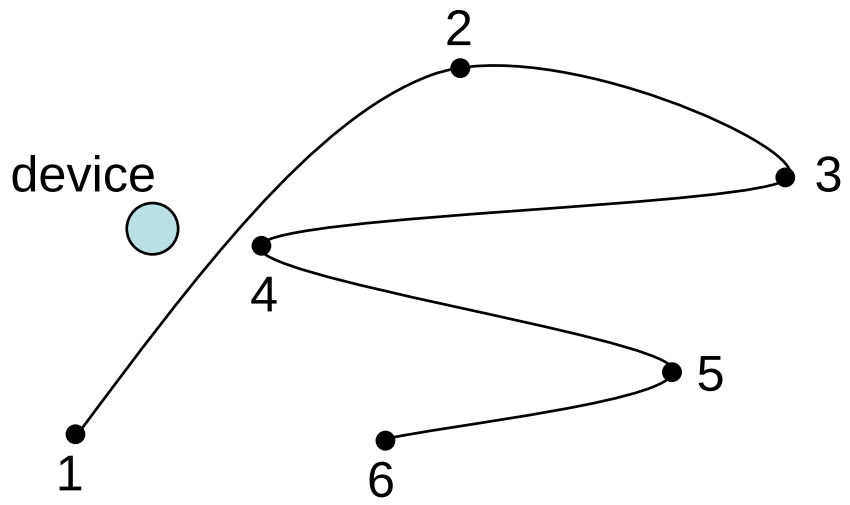



**Figure 45. The device should be constrained to the segment between vertices 1 and 2, but sparse sampling of the curve places it closer to vertex 4 than to either of these vertices. This motivates the use of a broader nearest-neighbor search to handle this case properly.**

```
// Get the points closest to the device
vector pdev = getDevicePosition();
curvePoint neighbors[N];
myKDTree.search(pdev, N, neighbors);
```

The N returned points are sorted by index, and for each returned point $p_i$ we build the two associated line segments ( $[p_{i-1} \rightarrow p_i]$ and $[p_i \rightarrow p_{i+1}]$ ) and insert them into an ordered list of candidate line segments that might contain the closest point to our haptic probe. This ordering reduces redundancy; we now have a maximum of (but generally less than) 2N line segments to search. We can compactly represent each line segment as its first index, so we can store the candidate set as an ordered, non-redundant list of indices:

```
// Sort the candidate line segments by index
std::set<unsigned int> candidateSegments;
for(unsigned int i=0; i<N; i++)
  candidateSegments.insert(neighbors[i]);
```

Now for each of those candidate segments, we compute the smallest distance between our device position $p_{dev}$ and the segment, using the approach presented (and available online) in [171]. We assume we have a function `distanceToSegment` that takes a test position and a segment defined by the indices of its two endpoints and returns the corresponding distance and point of closest approach (as a t-value, where 0.0 is the segment start and 1.0 is the segment endpoint). We find the segment with the smallest distance to the haptic device point:

```
// Find the point of closest approach among all candidate segments

struct distanceRecord {
  int segmentIdx;
  float t;
  float distance;
};

float shortestDistance = FLT_MAX;
distanceRecord closest;
std::set<unsigned int>::iterator iter;

// Loop over all candidate segments
for(iter=candidateSegments.begin();
  iter != candidateSegments.end(); iter++) {

  int index = *iter;
  float t;

  // What's the smallest distance to this segment?
  float distance =
    distanceToSegment(pdev,index,index+1,t);
  distanceRecord dr(index,t,distance);
```

```
  // Is this the smallest distance we've found so far (to any segment)?
  if (distance < shortestDistance) {
    closest = dr;
    shortestDistance = distance;
  }

}
```

For most curves, it is now sufficient to simply apply a constraint force pulling the device toward the closest point on the closest segment with stiffness $k_{constraint}$:

```
// Generate a constraint force pulling the haptic device toward the closest
// point on the curve

vector start =
  myCurve.points[closest.segmentIdx].pos;
vector end =
  myCurve.points[closest.segmentIdx +1].pos;
vector closestPoint =
  start + (end - start) * closest.t;
vector force =
  kconstraint * (closestPoint – pdev);
```

This approach, however, fails in the case illustrated in Figure 46. Here, due to normal deviation from the constraint path (resulting from limited stiffness), the device passes through vertex 4 on its way between vertices 1 and 2, but should still be constrained to segment [1,2] to guide the user along the correct curve shape. This can be handled by a modification to our distance-computation function, which takes into account the *arcdistance* of the point to which the haptic device was most recently constrained. Essentially, when choosing the closest point on the curve, we want to penalize points that are far from the test point both in Euclidean distance and in arclength.

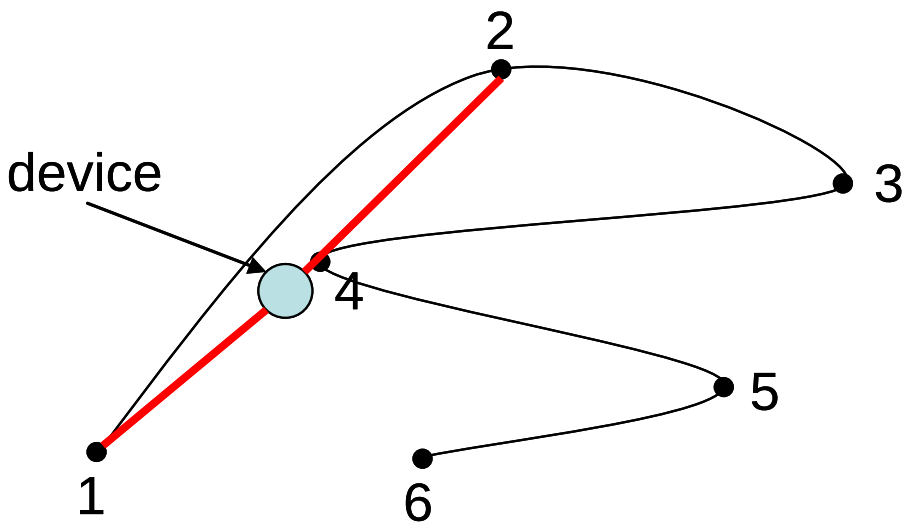



**Figure 46. The device passes through vertex 4 on its way between vertices 1 and 2, but should still be constrained to segment [1,2] to guide the user along the correct curve shape.**

We assume that the distance computation function is provided the arcposition of the point to which the device was previously constrained (or a flag indicating that this is a new constraint and there is no previous state, in which case the distance returned is just the usual Euclidean distance). For each segment we process, we find the closest point on that segment to the haptic device and compute the corresponding Euclidean distance as usual. We then take the absolute difference in arcposition between this point and the previous constraint point, multiply it by an empirically-selected penalty factor, and return this weighted "score" as our distance value in the above routine (this pseudocode replaces the distance computation in the above routine):

```
// known from our previous iteration
float previousArcPos;

float distance = distanceToSegment(pdev,
  index,index+1,t);

// Find the arcposition of the closest point of approach on this segment
float newArcPos =
  (myCurve.points[index].arcpos*t)
  +
```

```
  (myCurve.points[index+1].arcpos*(1.0-t));

// Find the arcdistance between this test point and my previous constraint
// position
float arcidst =
  fabs(previousArcPos - newArcPos);

// Weight our 'distance' value according to this arcposition.
distance = distance +
  arcidst * ARC_PENALTY_WEIGHT;
```

Higher values of ARC_PENALTY_WEIGHT maximally eliminate "jumping" along the curve (Figure 46). However, inappropriately high values may cause friction-like effects as the user rounds sharp corners in the curve and is prevented from "jumping" around corners when he *should* be allowed to move to subsequent segments. We have found this effect to be imperceptible for a wide range of values of ARC_PENALTY_WEIGHT (see Section 6.2.3).

### 6.2.3 Implementation and Results

The above algorithm was implemented in C++ using the public-domain kd-tree available in [136], a Phantom haptic device [110], and the CHAI 3D libraries for haptics and visualization [42]. Curves were generated according to [129], with 2000 points. N (number of nearest neighbors to search) was set to 100, with the arc penalty weight set to 1.0.

Figure 47 demonstrates the robustness of our approach. We see the actual path of the device in green, constrained by force vectors (indicated in black) to the curve. We see several regions (highlighted in blue) where the device very closely approaches a region of the curve that is distant from the current constraint position in terms of arclength, and the constraint position correctly remains on the current region of the curve.

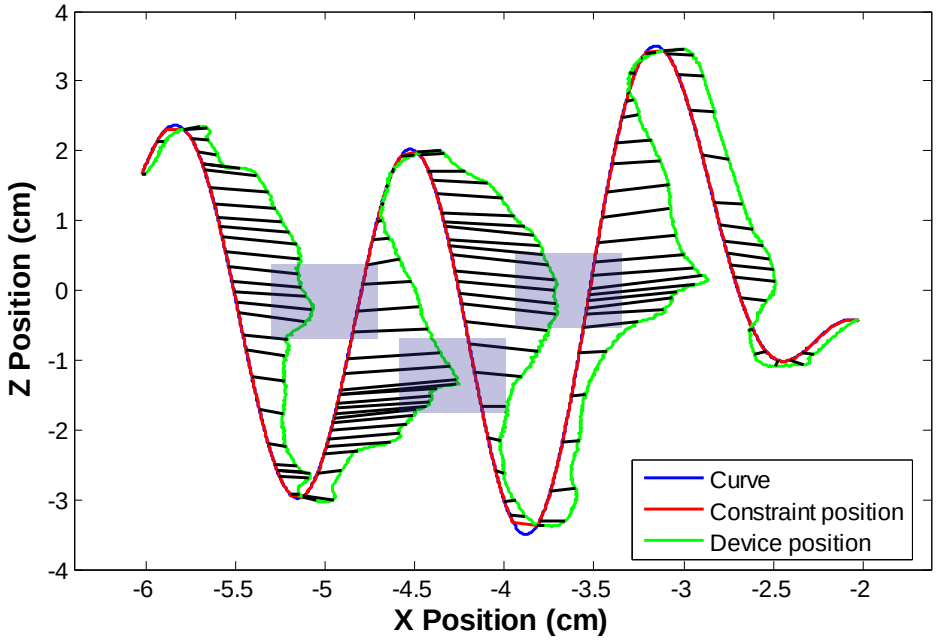



**Figure 47. Black lines indicate correspondences between device position (green) and constraint position (red). The highlighted areas show regions where the device approached a region on the curve that was distant in terms of arclength and was thus appropriately constrained to the current curve segment, despite being physically closer to the "incorrect" segment.**

For the constant values presented above, mean computation time per haptic iteration on a 1GHz Pentium 4 was 0.2ms, well below the accepted perceptual threshold of 1ms per haptic computation. Figure 48 shows the dependence of computation time on the number of samples in the trajectory for a fixed N (number of neighbors used in constraint search). We see that even with very large trajectories (up to two million samples), computation time is well below 1ms. Figure 49 shows the dependence of computation time on N (number of neighbors used in constraint search) for a fixed trajectory size. Although this increase is also approximately linear, there is a much more expensive constant factor associated with increasing N, since this increases the number of floating-point distance computations.

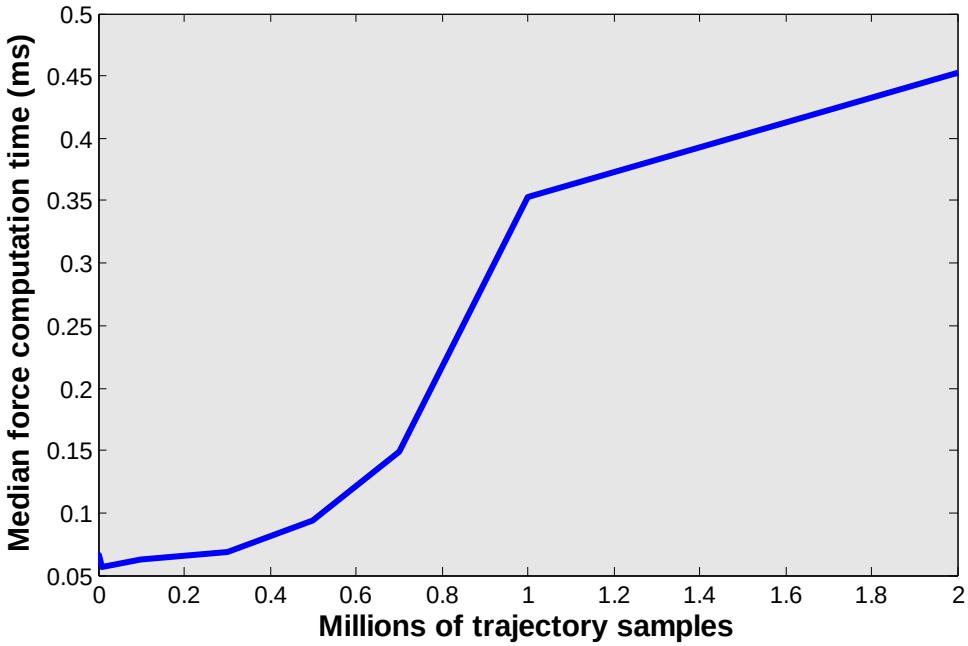



**Figure 48. Increase in computation time with increasing trajectory size, N (number of neighbors used) fixed at 100. The increase is approximately linear, but even with two million samples, the computation time is well under 1ms.**

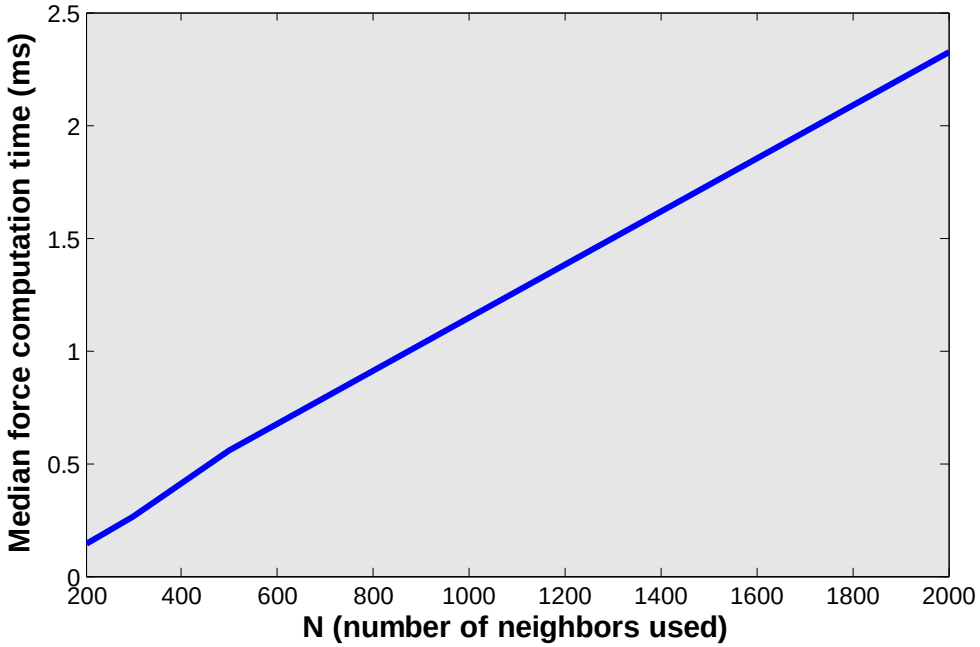


**Figure 49. Increase in computation time with increasing N (number of neighbors used), trajectory size fixed at 2000.**

As a final point, we note that our distance-computation function generates the closest point returned from each point/segment comparison, so this is the only part of our overall approach that would need to be modified to represent line segments as Bezier curve segments or other interpolation functions.

An implementation of the algorithm discussed here is included as part of our "haptic mentoring" experimental platform [129], available at:

http://cs.stanford.edu/~dmorris/haptic_training

## 6.3 Distance Map Generation

### 6.3.1 Terminology

For an object **O** on $\Re^n$ and a set of points **P** on $\Re^n$, the *distance field* is defined as the smallest distance from each point in **P** to a point on **O**. The distance metric is generally Euclidean distance, but any symmetric, non-negative function satisfying the triangle inequality can serve as a distance metric. The *distance map* is the distance field annotated with the *position* of the closest point on **O** for each point in **P**. When **O** is an orientable surface (a surface that partitions $\Re^n$ into two subspaces), the sign of the stored distance at a point indicates the subspace in which that point lies (in particular, this sign is often used to indicate whether a point is inside or outside a closed surface **O**). The *distance transform* takes a set of points **P** and an object **O** and annotates **P** with a distance map on **O**. The closely-related *closest-point transform* takes a set of points **P** and an object **O** and annotates each point in **P** with the location of the closest point on **O**, without distance information. The closest-point transform is computed by definition whenever a distance map is generated.

### 6.3.2 Background

The distance map is an implicit object representation with extensive applications in computer graphics, for example in physical simulation [59] and isosurface generation [205].

Distance maps have also been applied in haptics, to search for collisions between a haptic tool and the environment [117], to provide constraint forces when navigating a volume [17], and to constrain a surface contact point to the boundary of a region [98].

Several methods have been proposed for computing distance fields, distance maps, and closest point transforms. Many applications in computer animation use the approximate but extremely efficient Fast Marching Method [175]. [113] proposes a method based on Voronoi regions and local rasterization, and provides an open-source implementation [112]. More recently, approaches have emerged that use parallel graphics hardware to accelerate distance field computation [189].

We propose an alternative method for generating exact distance maps from point sets to triangle meshes that leverages bounding-box structures, which are already generated as a preprocessing step for many interactive applications in haptics and graphics.

### 6.3.3 Distance Map Generation

The following procedure assumes that we are given a list of points **P**, preferably sorted in an order that promotes spatial coherence (this is generally the case in practice, where regular voxel grids are used as the point set and sorting is trivial). We are also given a set of triangles **M**, which represent one or more logical objects in a scene.

We further assume that a bounding-volume hierarchy has been built on **M**. A strength of this approach is that it leverages common bounding-volume techniques, which are used in a variety of existing applications in haptics and graphics. Without loss of generality, we will assume that the hierarchy is composed of axis-aligned bounding boxes (AABB's). Further details on the construction and characteristics of AABB trees can be found in [40].

The general approach to finding the closest point on **M** to a point $\mathbf{P_i}$ in **P** is to descend the AABB tree, computing lower and upper bounds on the distance to each box we descend, and tracking the lowest upper bound $\mathbf{d_{lu}}$ we've encountered so far (the lowest "guaranteed" distance). If the lower bound for a box is farther from $\mathbf{P_i}$ than $\mathbf{d_{lu}}$, we can skip this box (see Figure 50). Using this culling approach and exploiting spatial coherence among subsequent points in **P** by selectively mixing breadth-first and depth-first examination of our bounding volume hierarchy, we can build distance maps in a manner that is both efficient and heavily parallelizable.

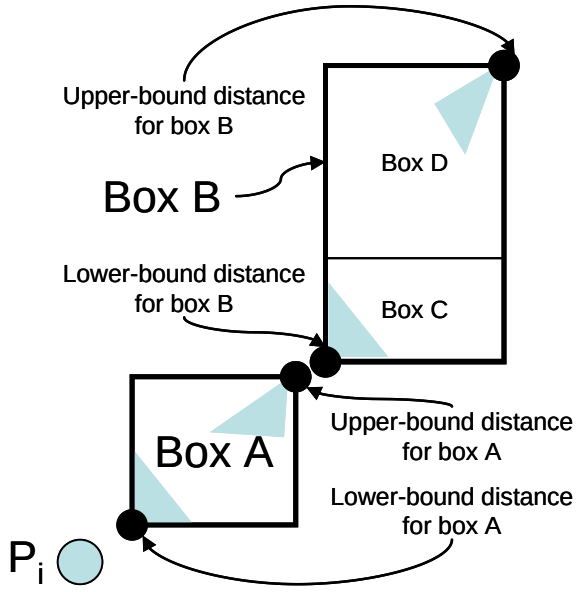



**Figure 50. Distance transformation for point $\mathbf{P_i}$. If we've processed Box A before we process Box B, we will not descend to Box B's children, because Box B's lower-bound distance is greater than Box A's upper-bound distance.**

In the following pseudocode, we assume without loss of generality that the AABB tree representing our triangle mesh is in the same coordinate frame as our point list; in practice coordinate transformations are performed before distance computation begins. We also assume for clarity of terminology that the list of points **P** is a series of voxel locations (this is the case when computing the distance transform on a regular grid), so we refer to the $\mathbf{P_i}$'s as "voxels" and locations on the surface **M** as "points".

```
// A simple AABB tree hierarchy
```

```
// A generic tree node maintaining only a parent pointer.  This pseudocode
// avoids pointer notation; all links within the tree and all references to
// AABBNode's in the code should be read as pointers.
struct AABBNode { AABBNode parent; };

// A structure representing a bounding box and pointers to child nodes.
struct AABBox : public AABBNode {

  // the actual bounding box
  vector3 xyzmax, xyzmin;

  // my children in the AABB tree
  AABBNode left, right;
}

// A structure representing a leaf node
struct AABBLeaf : public AABBNode {
  triangle t;
}

// The inputs to our problem

// The P_i's
std::list<vector3> voxels;

// The triangle set M, pre-processed into an AABB tree
AABBox tree_root;

// All the boxes we still need to look at for the current voxel.  This may
// not be empty after a voxel is processed; placing nodes here to be used for
// the next voxel is our mechanism for exploiting spatial coherence.
std::list<AABBNode> boxes_to_descend;

// The smallest squared distance to a triangle we've seen so far for the
// current voxel...
//
// We generally track squared distances, which are faster to compute than
// actual distances.  When all is said and done, taking the square root of
// this number will give us our distance value for this voxel.
float lowest_dist_sq = FLT_MAX;

// The point associated with this distance
vector3 closest_point;

// The tree node associated with the closest point.  We store this to help us
// exploit spatial coherence when we move on to our next voxel.
//
// This will always be a leaf.
AABBNode closest_point_node;

// The lowest upper-bound squared distance to a box we've seen so far for
// the current voxel.
float lowest_upper_dist_sq = FLT_MAX;

// Process each voxel on our list, one at a time...

std::list<vector3>::iterator iter =
  voxels.begin();

while (iter != voxels.end) {
```

```
  // Grab the next point
  vector3 v = (*iter);

  // Now we're going to find the closest point in the tree (tree_root)
  // to v...
  //
  // See below for the implementation of find_closest_point.
  find_closest_point(v);

  // Now output or do something useful with lowest_dist_sq and closest_point;
  // these are the values that should be associated with v in our output
  // distance map...
  do_something_useful();

  // So it's time to move on to the next voxel.  We'd like to exploit spatial
  // coherence by giving the next voxel a "hint" about where to start looking
  // in the tree.  See the explanation below for what this does; the summary
  // is that it seeds 'boxes_to_descend' with a good starting point for the
  // next voxel.
  seed_next_voxel_search();

}

// Find the closest point in our mesh to the sample point v
void find_closest_point(vector3 v) {

  // Start with the root of the tree
  boxes_to_descend.push_back(tree_root);

  while(!(boxes_to_descend.empty)) {
    AABBNode node =
      boxes_to_descend.pop_front();
    process_node(node,v);
  }

}

// Examine the given node and decide whether we can discard it or whether we
// need to visit his children.  If it's a leaf, compute an actual distance
// and store it if it's the closest so far.
//
// Used as a subroutine in the main voxel loop (above).
void process_node(AABBNode node, vector3 v){

  // Is this a leaf?  We assume we can get this from typing, or that the
  // actual implementation uses polymorphism and avoids this check.
  bool leaf = isLeaf(node);

  // If it's a leaf, we have no more descending to do, we just need to
  // compute the distance to this triangle and see if it's a winner.
  if (leaf) {

    // Imagine we have a routine that finds the distance from a point to a
    // triangle; [171] provides an optimized routine with a thorough
    // explanation.
    float dsq;
    vector3 closest_pt_on_tri;

    // Find the closest point on our triangle (leaf.t) to v, and the squared
    // distance to that point.
    compute_squared_distance(v,leaf.t,
      dsq,closest_pt_on_tri;
```

```
  // Is this the shortest distance so far?
  if (dsq < lowest_dist_sq) {

    // Mark him as the closest we've seen
    lowest_dist_sq = dsq;
    closest_point = clost_pt_on_tri;
    closest_point_node = node;

    // Also mark him as the "lowest upper bound", because any future boxes
    // whose lower bound is greater than this value should be discarded.
    lowest_upper_dist_sq = dsq;
  }

  // This was a leaf; we're done with him whether he was useful or not.
  return;
}

// If this is not a leaf, let's look at his lower- and upper-bound
// distances from v.
//
// Computing lower- and upper-bound distances to an axis-aligned bounding
// box is extremely fast; we just take the farthest plane on each axis
float best_dist = 0;
float worst_dist = 0;

// If I'm below the x range, my lowest x distance uses the minimum x, and
// my highest uses the maximum x
if (v.x < node.box.xyzmin.x) {
  best_dist += node.box.xyzmin.x - v.x;
  worst_dist += node.box.xyzmax.x - v.x;
}

// If I'm above the x range, my lowest x distance uses the maximum x, and
// my highest uses the minimum x
else if (v.x > node.box.xyzmax.x) {
  best_dist += v.x - node.box.xyzmax.x;
  worst_dist += v.x - node.box.xyzmin.x;
}

// If I'm _in_ the x range, x doesn't affect my lowest distance, and my
// highest-case distance goes to the _farther_ of the two x distances
else {
  float dmin =
     fabs(node.box.xyzmin.x - v.x);
  float dmax =
     fabs(node.box.xyzmax.x - v.x);
  double d_worst = (dmin>dmax)?dmin:dmax;
  worst_dist += d_worst;
}

// Repeat for y and z...

// Convert to squared distances
float lower_dsq = best_dist * best_dist;
float upper_dsq = worst_dist * worst_dist;

// If his lower-bound squared distance is greater than
// lowest_upper_dist_sq, he can't possibly hold the closest point, so we
// can discard this box and his children.
if (lower_dsq > lowest_upper_dist_sq)
  return;
```

```
  // Check whether I'm the lowest upper-bound that we've seen so far,
  // so we can later prune away non-candidate boxes.
  if (upper_dsq < lowest_upper_dist_sq) {
    lowest_upper_dist_sq = upper_dsq;
  }

  // If this node _could_ contain the closest point, we need to process his
  // children.
  //
  // Since we pop new nodes from the front of the list, pushing nodes to the
  // front here results in a depth-first search, and pushing nodes to the
  // back here results in a breadth-first search.  A more formal analysis of
  // this tradeoff will follow in section 6.3.4.
  boxes_to_descend.push_front(node.left);
  boxes_to_descend.push_front(node.right);

  // Or, for breadth-first search...
  // boxes_to_descend.push_back(node.left);
  // boxes_to_descend.push_back(node.right);
}
```

When we've finished a voxel and it's time to move on to the next voxel, we'd like to exploit spatial coherence by giving the next voxel a "hint" about where to start looking in the tree. We expect the node that contains the closest point to the next voxel to be a "near sibling" of the node containing the closest point to the current voxel, so we'll let the next voxel's search begin at a nearby location in the tree by walking a couple nodes up from the best location for this voxel.

The constant TREE_ASCEND_N controls how far up the tree we walk to find our "seed point" for the next voxel. Higher values assume less spatial coherence and require more searching in the case that the next voxel is extremely close to the current voxel. Lower values assume more spatial coherence and optimize the case in which subsequent voxels are very close, while running a higher risk of a complete search.

Section 6.3.4 discusses the selection of an optimal value for TREE_ASCEND_N.

```
void seed_next_voxel_search() {

  // Start at the node that contained our closest point and walk a few levels
  // up the tree.
  AABBNode seed_node = closest_point_node;
  for(int i=0; i<TREE_ASCEND_N; i++) {
    if (seed_node.parent == 0) break;
    else seed_node = seed_node.parent;
  }

  // Put this seed node on the search list to be processed with the next
```

```
  // voxel.
  boxes_to_descend.push_back(seed_node);

}
```

In summary, for each voxel in $\mathbf{P_i}$ we track the lowest upper-bound distance that we've found for a box as we descend our AABB tree, and discard boxes whose lower-bound distance is larger. When we reach a leaf node, we explicitly compute distances and compare to the lowest distance we found so far. We exploit spatial coherence when processing a voxel by first searching a small subtree in which we found the closest point for the previous voxel.

### 6.3.4 Implementation and Results

The approach presented here was evaluated in the context of generating internal distance fields (finding and processing only voxels that lie inside a closed mesh) during the process of voxelization. *Voxelizer* is an application written in C++ that loads meshes and uses a flood-filling process to generate voxel representations of those meshes, optionally including distance fields. Both the flood-filling and the distance-field generation use the public-domain AABB tree available in CHAI [42].

To evaluate the suitability of our approach and the benefit of our exploitation of spatial coherence, we generated voxel arrays and distance fields for a variety of meshes (Figure 51 and Figure 52) at a variety of voxel densities and a variety of values for TREE_ASCEND_N (see above). Furthermore, at each parameter set, we generated distance fields using both depth- and breadth-first search. The following sections discuss the performance results from these experiments.

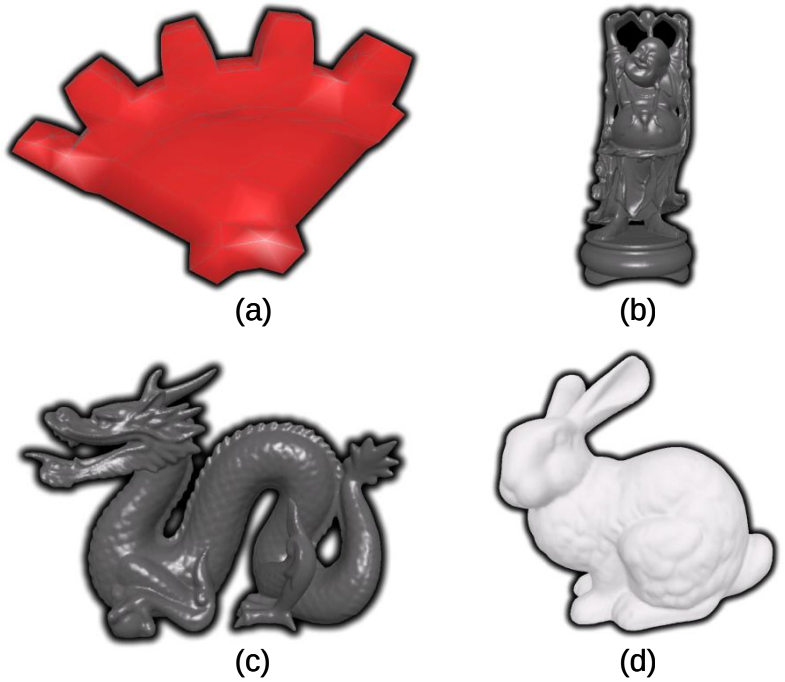



**Figure 51. Meshes used for evaluating distance map computation. (a) Gear: 1000 triangles. (b) Happy: 16000 triangles. (c) Dragon: 203000 triangles (d) Bunny: 70,000 triangles.**

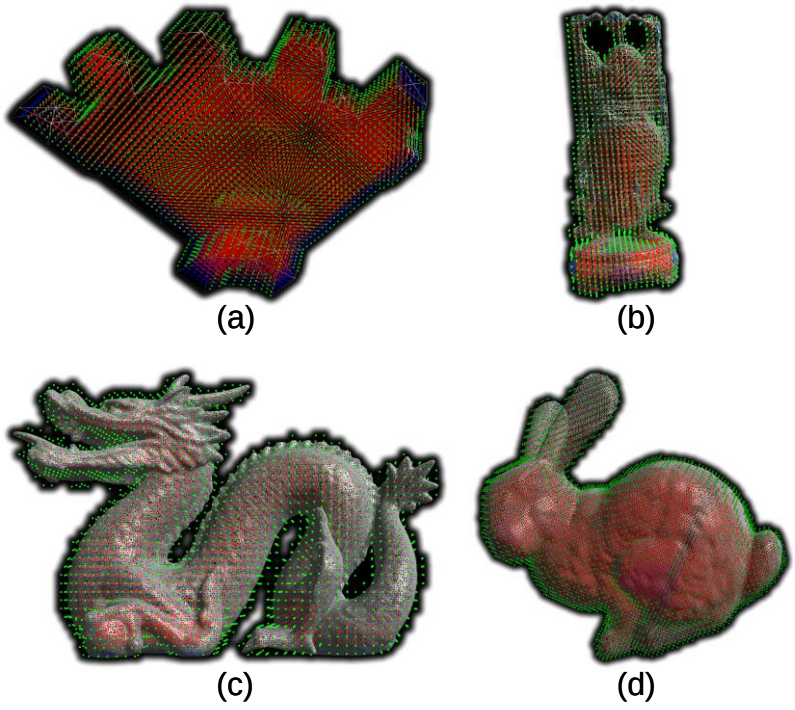


**Figure 52. The same meshes displayed in Figure 51, after using the *voxelizer* application to identify internal voxels (voxel centers are shown in green for surface voxels and red for internal voxels) by flood-filling. The long axis resolution in each case here is 50 voxels.**

### 6.3.4.1 Overall Performance

Table 3 shows the computation time for flood-filling and distance-field generation for each of the four test meshes at a variety of resolutions. The "dragon", "bunny", and "happy" models were obtained from the Stanford 3D Scanning Repository [82]. The "gear" model was obtained from the TetGen examples page [83]. The voxel arrays generated contain surface and internal voxels only; the full distance field for voxels outside the mesh is not generated. "Long axis resolution" indicates the number of voxels into which the longest axis of the mesh's bounding-box is divided; voxels are isotropic so the resolutions of the other axes are determined by this value.

| Mesh | Triangles | Long axis | Voxels | Total time (s) | Distance time (s) |
|---|---|---|---|---|---|
| bunny | 70k | 30 | 7168 | 0.736 | 0.683 |
| bunny | 70k | 75 | 95628 | 6.107 | 5.282 |
| bunny | 70k | 135 | 529024 | 29.033 | 25.258 |
| bunny | 70k | 195 | 1561728 | 82.341 | 71.585 |
| gear | 1k | 30 | 4156 | 0.144 | 0.117 |
| gear | 1k | 75 | 54270 | 1.751 | 1.383 |
| gear | 1k | 135 | 286813 | 9.228 | 7.282 |
| gear | 1k | 195 | 829321 | 27.137 | 21.387 |
| happy | 16k | 30 | 2020 | .13495 | .1177 |
| happy | 16k | 75 | 25308 | 1.387 | 1.208 |
| happy | 16k | 135 | 132910 | 6.132 | 5.261 |
| happy | 16k | 195 | 381120 | 16.956 | 14.48 |
| dragon | 203k | 30 | 2550 | 0.494 | 0.47 |
| dragon | 203k | 75 | 31674 | 3.158 | 2.859 |
| dragon | 203k | 135 | 164061 | 11.839 | 10.558 |
| dragon | 203k | 195 | 468238 | 30.13 | 26.633 |

**Table 3. A comparison of flood-filling and distance-computation times for all four meshes at a variety of voxel resolutions.**

We note that for small resolutions, on the order of 30 voxels, times for distance computation are interactive or nearly interactive, even for complex meshes. We also note that in general, distance computation represents the significant majority of the total time required to perform the combined flood-filling and distance-field generation (on average, distance-field generation represents 86% of the total time).

Figure 53 shows the dependence of computation time on long axis resolution for all four meshes. As expected, all meshes display an exponential increase in computation time as voxel resolution increases, but even at very high resolutions, computation time is tractable for preprocessing applications (only above one minute for one of the four meshes and only above a long axis resolution of 180 voxels).

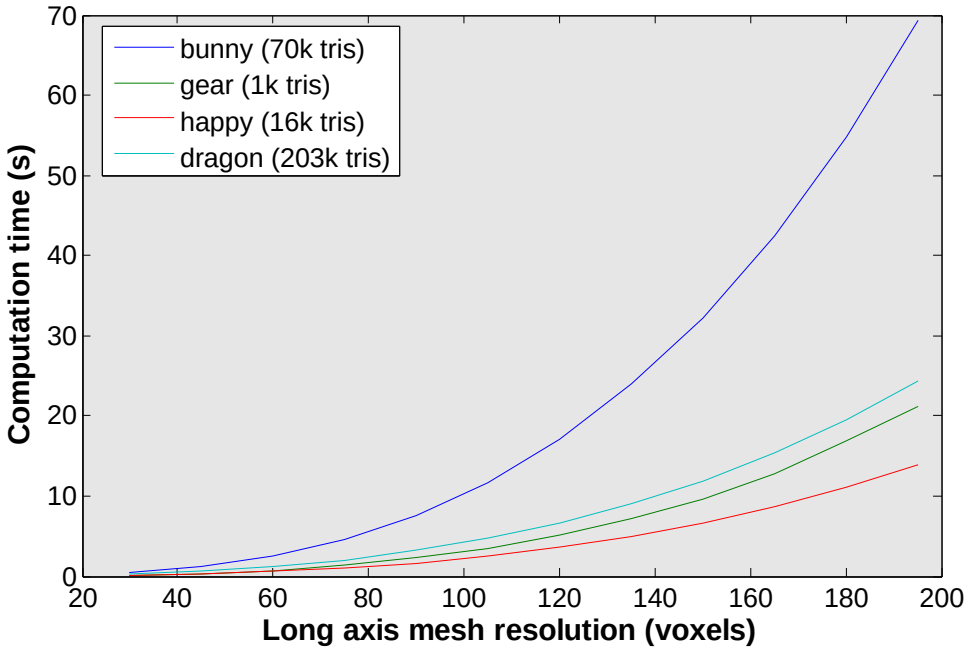



**Figure 53. Performance of our distance-map computation approach on all four meshes at a variety of mesh resolutions.**

### 6.3.4.2 Spatial Coherence

To analyze the benefit of exploiting spatial coherence in distance-map computation, and to identify the optimal value of TREE_ASCEND_N (the number of tree levels to step up in generating a "hint" location for the next voxel's distance search), voxel arrays and distance fields were generated for all four meshes with various values of TREE_ASCEND_N. Figure 54 shows the results for the "happy" mesh (this mesh was chosen arbitrarily; results were similar for all four meshes). A TREE_ASCEND_N value of -1 indicated that spatial coherence was not exploited at all; i.e. every distance search started at the top of the tree. A value of 0 indicated that the "hint" node was the *leaf* node (a single triangle) that contained the shortest distance for the previous voxel.

Exploting spatial coherence yields five-fold improvement in performance (a reduction in distance field time from 62 seconds to 13 seconds) for the largest resolution shown in Figure 54. This corresponds to the difference between TREE_ASCEND_N values of 0 and 1. Further increasing TREE_ASCEND_N does not further improve performance; it is clear in Figure 54 that zero is the optimal value. This is equivalent to assuming that locality extends as far as the closest triangle; it isn't worth searching neighboring AABB nodes as well before searching the whole tree.

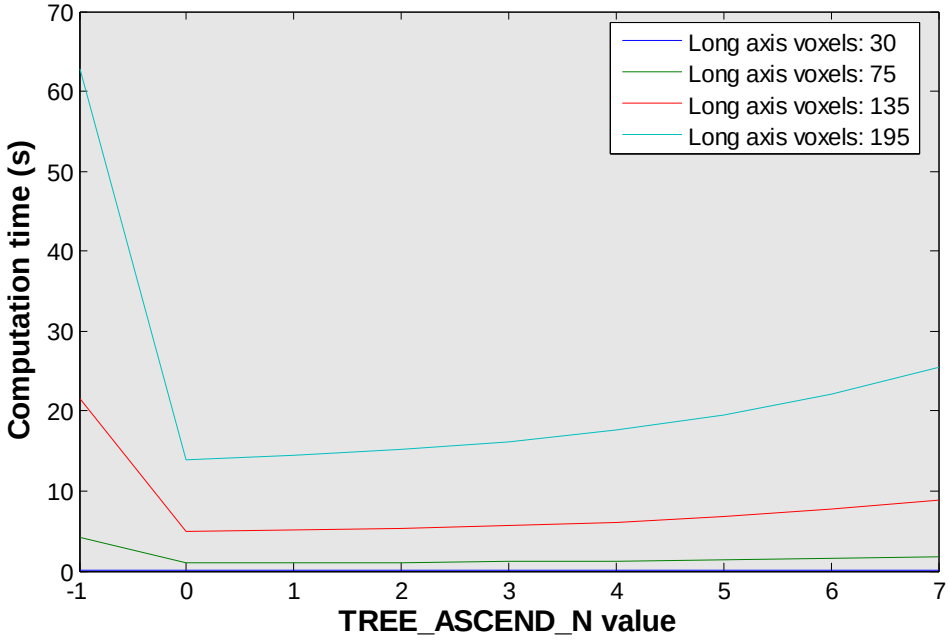



**Figure 54. Performance benefit of exploiting spatial coherence and optimal value selection for TREE_ASCEND_N (results shown here are for the "happy" mesh). A value of -1 indicated that spatial coherence was not exploited at all. A value of 0 indicated that the "hint" node was the leaf node (a single triangle) that contained the shortest distance for the previous voxel.**

### 6.3.4.3 Depth- vs. Breadth-First Search

To compare the use of depth- and breadth-first distance search, voxel arrays and distance fields were generated for all four meshes using each approach. Figure 55 shows the results when using the optimal TREE_ASCEND_N value of 0. Depth-first search is consistently better, but by a very small margin.

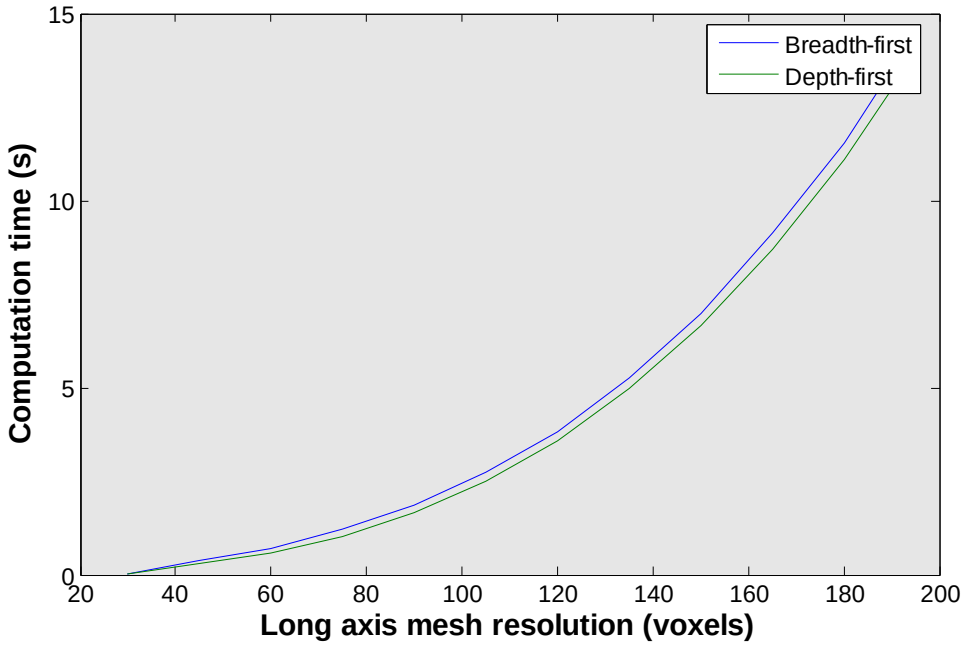



**Figure 55. Comparison of depth- and breadth-first search for the "happy" mesh using a TREE_ASCEND_N value of 0 (optimal).**

When spatial coherence is not exploited – which serves as a surrogate for the case in which the point set is not sorted and does not provide strong spatial coherence – depth-first search performs significantly better. This is illustrated in Figure 56, which shows results for the "happy" mesh at various resolutions with no assumption of spatial coherence.

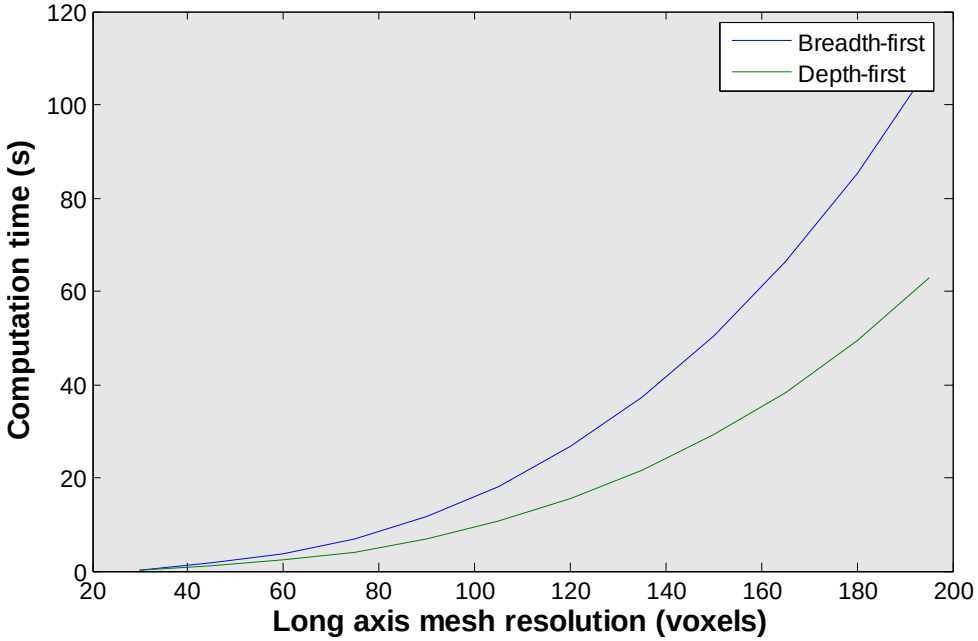



**Figure 56. Comparison of depth- and breadth-first search for the "happy" mesh using a TREE_ASCEND_N value of -1 (no exploitation of spatial coherence).**

### 6.3.5 Implementation Availability

A binary version of this application, with documentation and the models used in these experiments, is available online at:

http://cs.stanford.edu/~dmorris/voxelizer

*Voxelizer* is currently used to generate the voxel meshes used in [126]; distance fields are used to shade voxels based on their distances to anatomic structures.

Future work will include leveraging the obvious parallelism available in this approach; voxels are processed nearly independently and could easily be distributed across machines with a nearly linear speedup. Furthermore, the simple nature of the computations performed here makes this suitable to parallelization across simple processing units, such as those available on commercial GPU's, which have been successfully used to process AABB-based collision queries by [201]. We would also

like to explore the performance impact of using other bounding-volume hierarchies (e.g. oriented-bounding-box trees and sphere trees), which fit trivially into our framework.

## 6.4 Haptic Data Logging

### 6.4.1 Background

It is conventionally accepted that a user will begin to notice discretization artifacts in a haptic rendering system if the system's update rate falls below 1kHz. Furthermore, as a haptic application's update rate falls, the system becomes more prone to instability and constraint violation. With this in mind, it is essential that designers of haptic software structure applications to allow high-bandwidth, low-latency generation of haptic forces.

There are two relevant implications of this requirement. First of all, haptic computation must run on a thread that allows computation at 1kHz. This is non-trivial on single-CPU systems running non-real-time operating systems, which typically have thread timeslices of 15ms or more. In other words, naively sharing the CPU among a haptic application thread and other application or system threads will not nearly provide the necessary performance. Boosting thread and process priority is a simple solution that is offered by common OS's, but indiscriminately boosting thread priority can prevent other application tasks (e.g. graphic rendering) and even critical operating system services from running. Common solutions to this problem include using dual-CPU PC's, boosting thread priority while manually ensuring that the persistent haptic loop will yield periodically, and/or using hardware-triggered callbacks to control the rate of haptic force computation.

Additionally, this stringent performance constraint means that "slow" tasks (those that require more than one millisecond on a regular basis) cannot be placed in the critical path of a haptic application. Graphic rendering, for example, is often computationally time-consuming and is generally locked to the refresh rate of the display, allowing a peak throughput of approximately 30Hz on most systems (lower if

the graphical scene is particularly complex). For this reason, nearly all visuohaptic applications decouple graphic and haptic rendering into separate threads.

Disk I/O is another task that incurs high latencies (often over 10ms), particularly when bandwidth is high. For a haptic application that requires constantly logging haptic data to disk – such as a psychophysical experiment involving a haptic device – it is essential to place blocking disk I/O on a thread that is distinct from the haptic rendering thread.

Using this common scheme, data synchronization between a haptic thread (which collects position data from the haptic device, computes forces, and sends forces to the device) and a "slow" thread (handling graphics and disk I/O) can become a bottleneck. Traditional locks allow the slow thread to block the haptic thread, and if the locked region includes a high-latency operation, the haptic thread can stall for an unacceptable period. Many applications are able reduce the data exchanged among threads to a few vectors or small matrices, and forego synchronization entirely since the probability and impact of data conflicts are rare.

Data logging tasks, however, cannot take this approach. Even small errors resulting from race conditions can place data files in an unrecoverable state. Furthermore, the high bandwidth of data flow increases the probability of conflicts if data queued for file output is stored in a traditional linked list. We thus present a data structure that allows lock-free synchronization between a producer thread and a consumer thread, with the constraint that the consumer thread does not need to access data immediately after the data are produced. The only synchronization primitive required is an atomic pointer-sized write, provided by all current hardware. This structure does not address sleeping; it's assumed that the producer never sleeps (it's a high-priority loop). Periodically waking the consumer – who might sleep – is a trivial extension.

We present this approach in the context of a haptic application, but it's equally applicable to other applications with similar threading structures, for example neurophysiological and psychophysical experiments. For example, the

implementation discussed here is used by the software presented in [131], which is used in the experiments presented in [145].

### 6.4.2 Data Structure

The data structure presented is labeled a "blocked linked list" (BLL). The BLL is a linked list of blocks of data records; the list's head pointer is manipulated only by the consumer, and the list's tail pointer is manipulated only by the producer. The BLL is initialized so that the head and tail pointers point to a single block. In pseudocode:

```
struct bll_record {
  // the relevant data structure is defined here; in practice the BLL is
  // templated and this structure is not explicitly defined
};

struct bll_block {

  // the data stored in this block
  bll_record data[BLOCK_SIZE];

  // how many data records have actually
  // been inserted?
  int count=0;

  // conventional linked list next pointer
  bll_block* next=0;

};

struct BLL {

  // conventional linked list head/tail ptrs
  bll_block *head,*tail;

  // initialize to a new node
  BLL() { head = tail = new bll_block; }

};
```

The BLL offers the following interface:

```
// This function is called only by the producer (haptic) thread to insert a
// new piece of data into the BLL.
void BLL::push_back(bll_record& d) {

  // If we've filled up a block,  allocate a new one.  There's no risk of
  // conflict because the consumer never accesses the tail.
  if (tail->count == BLOCK_SIZE) {

    bll_block* newtail = new bll_block;
    newtail->next = tail;
```

```
    // After this, I can never touch the old tail again, since the consumer
    // could be using it.
    tail = newtail;

  }

  // insert the new data record
  tail->data[count] = d;
  count++;

}

// This function is called only by the consumer (logging) thread to flush all
// available data to disk
void BLL::safe_flush() {

  // If the tail pointer changes during this call, after this statement,
  // that's fine; I'll only log up to  the tail at this instant.  I can't
  // access 'tail' directly for the rest of this call.
  bll_block* mytail = tail;

  // If there are no filled blocks, this loop won't run; no harm done.
  while(head != mytail) {

    // Dump this whole block to disk or perform other high-latency operations
    fwrite(head->data, sizeof(bll_record),BLOCK_SIZE,myfile);

    // Increment the head ptr and clean up what we're done with
    bll_block oldhead = head;
    head = head->next;
    delete oldhead;

  }

};
```

The central operating principle is that the `push_back` routine only accesses the current tail; when the tail is filled, a new block becomes the tail and this routine never touches the old tail again. The `safe_flush` routine flushes all blocks *up to but not including* the current tail. If the current tail changes during this routine's execution, it may leave more than one block unflushed, but it will not conflict with the producer's `push_back` routine.

These two routines comprise the important components of the data structure; required but not detailed here are additional initialization routines and a "tail flush" routine that flushes the current tail block and can be run when the producer is permanently finished or has downtime (the pseudocode above never flushes the last, partially-filled block). The BLL also presents an O(N) routine for safe random

element access by the consumer thread, allowing access to elements up to but not including the head block.

### 6.4.3 Implementation and Results

A template-based, C++ implementation of this data structure is available at:

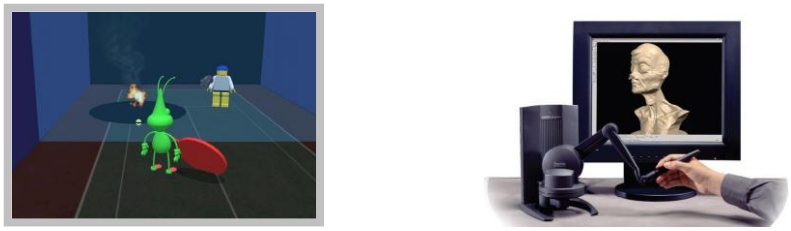
http://cs.stanford.edu/~dmorris/code/block_linked_list.h

This implementation was used in [129], [131], and [145], and introduced no disk latency on the high-priority haptic/experiment threads.

`BLOCK_SIZE` is a performance variable; in practice it is also templated but it need not be the same for every block. Higher values improve bandwidth on the consumer thread, since larger disk writes are batched together and allocated memory is more localized, but may result in larger peak latencies on the consumer thread (due to larger writes). Higher values of `BLOCK_SIZE` also increase the latency between production and consumption. A BLOCK_SIZE value of 1000 was used in [129], [131], and [145].

# 7 Standardized Evaluation of Haptic Rendering Systems

For many applications in haptics, haptic feedback doesn't need to be accurate in a physical sense, it just needs to "feel good" in a way that's appropriate for the application. This includes, for example, haptic games (Figure 57a) and haptic CAD applications like virtual sculpting (Figure 57b).



(a) (b)

**Figure 57. Haptic applications that require only limited force-realism.**

But for applications that require transferring skills from a virtual environment to the real world, in particular for our haptic surgical training system, it's critical that a haptic training environment not only be usable, but also be haptically accurate in terms of the absolute forces that the user is learning to apply in simulation. However, haptic applications are often built using existing algorithms and libraries whose real-world accuracy has not been experimentally verified. This is a problem that faces us in developing and verifying the surgical simulation environment presented in Section 3.

On a related note, haptics researchers and engineers face a particular challenge in reporting their results and assessing published results. The bottom line result in many papers on haptics is that an author has devised a new method for rendering some physical phenomenon (e.g. friction, texture, etc.) but, it is often difficult to convey or assess the true result – which is a haptic sensation – in a printed or electronic publication.


So we'd like to address both of these problems: assessing the absolute accuracy of haptic rendering algorithms, and also providing a standardized way of presenting and comparing results in haptics. This section of the thesis describes the use of haptic ground truth data to approach these challenges.

The work presented here was published as [165].

---

The development and evaluation of haptic rendering algorithms presents two unique challenges. Firstly, the haptic information channel is fundamentally bidirectional, so the output of a haptic environment is fundamentally dependent on user input, which is difficult to reliably reproduce. Additionally, it is difficult to compare haptic results to real-world, "gold standard" results, since such a comparison requires applying identical inputs to real and virtual objects and measuring the resulting forces, which requires hardware that is not widely available. We have addressed these challenges by building and releasing several sets of position and force information, collected by physically scanning a set of real-world objects, along with virtual models of those objects. We demonstrate novel applications of this data set for the development, debugging, optimization, evaluation, and comparison of haptic rendering algorithms.

## 7.1 Introduction and Related Work

Haptic rendering systems are increasingly oriented toward representing realistic interactions with the physical world. Particularly for simulation and training applications, intended to develop mechanical skills that will ultimately be applied in the real world, fidelity and realism are crucial.

A parallel trend in haptics is the increasing availability of general-purpose haptic rendering libraries (e.g. [42], [174], [173]), providing core rendering algorithms that can be re-used for numerous applications. Given these two trends, developers and users would benefit significantly from standard verification and validation of haptic rendering algorithms.

In other fields, published results often "speak for themselves" – the correctness of mathematical systems or the realism of images can be validated by reviewers and peers. Haptics presents a unique challenge in that the vast majority of results are fundamentally interactive, preventing consistent repeatability of results. Furthermore, it is difficult at present to distribute haptic systems with publications, although several projects have attempted to provide deployable haptic presentation systems ([42], [70]).

Despite the need for algorithm validation and the lack of available approaches to validation, little work has been done in providing a general-purpose system for validating the physical fidelity of haptic rendering systems. Kirkpatrick and Douglas [99] present a taxonomy of haptic interactions and propose the evaluation of complete haptic systems based on these interaction modes, and Guerraz et al [71] propose the use of physical data collected from a haptic device to evaluate a user's behavior and the suitability of a device for a particular task. Neither of these projects addresses realism or algorithm validation. Raymaekers et al [159] describe an objective system for comparing haptic algorithms, but do not correlate their results to real-world data and thus do not address realism. Hayward and Astley [78] present standard metrics for evaluating and comparing haptic devices, but address only the physical devices and do not discuss the software components of haptic rendering systems. Similarly, Colgate and Brown [41] present an impedance-based metric for evaluating haptic devices. Numerous projects (e.g. [58], [193]) have evaluated the efficacy of specific

haptic systems for particular motor training tasks, but do not provide general-purpose metrics and do not address realism of specific algorithms. Along the same lines, Lawrence et al [104] present a perception-based metric for evaluating the maximum stiffness that can be rendered by a haptic system.

This section addresses the need for objective, deterministic haptic algorithm verification and comparison by presenting a publicly available data set that provides forces collected from physical scans of real objects, along with polygonal models of those objects, and several analyses that compare and/or assess haptic rendering systems. We present several applications of this data repository and these analysis techniques:

- Evaluation of rendering realism: comparing the forces generated from a physical data set with the forces generated by a haptic rendering algorithm allows an evaluation of the physical fidelity of the algorithm.

- Comparison of haptic algorithms: Running identical inputs through multiple rendering algorithms allows identification of the numeric strengths and weaknesses of each.

- Debugging of haptic algorithms: identifying specific geometric cases in which a haptic rendering technique diverges from the correct results allows the isolation of implementation bugs or scenarios not handled by a particular approach, independent of overall accuracy.

- Performance evaluation: Comparing the computation time required for the processing of a standard set of inputs allows objective comparison of the performance of specific implementations of haptic rendering algorithms.

The data and analyses presented here assume an impedance-based haptic rendering system and a single point of contact between the haptic probe and the object of

interested. This work thus does not attempt to address the full range of possible contact types or probe shapes. Similarly, this work does not attempt to validate the realism of an entire haptic rendering pipeline, which would require a consideration of device and user behavior and perceptual psychophysics. Rather, we present a data set and several analyses that apply to a large (but not universal) class of haptic rendering systems. We leave the extension of this approach to a wider variety of inputs and to more sophisticated metrics as future work.

The remainder of this paper is structured as follows: Section 7.2 will describe our system for physical data acquisition, Section 7.3 will describe the process by which we simulate a contact trajectory for evaluation of a haptic rendering algorithm, Section 7.4 will describe some example results we have obtained through this process, and Section 7.5 will discuss the limitations of our method and several scenarios in which our data and methods may be useful to others in the haptics community. We conclude with a description of our public data repository and a discussion of future extensions to this work.

## 7.2 Data Acquisition

Haptic rendering algorithms typically have two sources of input: a geometric model of an object of interest and real-time positional data collected from a haptic interface. The output of this class of algorithms is typically a stream of forces that is supplied to a haptic interface. A key goal of our data and analyses is to compare this class of algorithms to real-world data, which requires: (a) collecting or creating a geometric model of a real-world object and (b) collecting a series of correlated forces and positions on the surface of that object.

We have constructed a sensor apparatus that allows the collection of this data. Our specific goal is to acquire data for haptic interaction with realistic objects using a hand-held stylus or pen-like device (henceforth called “the probe”). We use the HAVEN, an integrated multisensory measurement and display environment at Rutgers, for acquiring measurements interactively, with a human in the loop.

In previous work ([147], [148]), we acquired such measurements using a robotic system called ACME (the UBC Active Measurement facility). This robotic approach has many advantages, including the ability to acquire repeatable and repetitive measurements for a long period of time, and the ability to acquire measurements from remote locations on the Internet. However, our current goals are different, and a hand-held probe offers a different set of advantages that are important for evaluating interaction with a haptic device.

First, it measures how a real probe behaves during natural human interaction, and therefore provides more meaningful data for comparison. This is important, because contact forces depend in part on the passive, task-dependent impedance of the hand holding the probe, which is difficult to measure or to emulate with a robot arm. Second, the dexterity of robot manipulators available today is very poor in comparison with the human hand. Furthermore, acquiring measurements in concave regions or near obstacles using a robot is very difficult, but is easy for a human.

We acquired three types of measurements for each object in our data repository:

1. The object's 3D shape
2. Motion of the probe tip relative to the object
3. The force on the probe tip during contact

We describe these measurements in the remainder of this section, in reverse order. Force data are acquired using a custom-designed hand-held probe built around a Nano17 6-axis force/torque sensor (Figure 58) (ATI Industrial Automation, Apex, NC, USA). The reported spatial resolution of the force sensor is as follows (the z-axis is aligned with the axis of the probe): Fx,Fy 1/320 N; Fz 1/640 N; Tx,Ty 1/128 N·mm; Tz 1/128 N·mm.

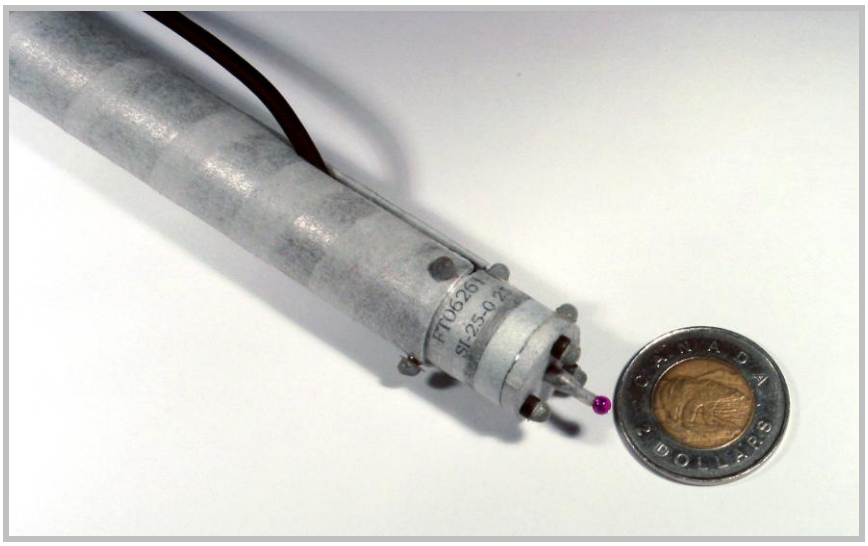


**Figure 58. The sensor used to acquire force and torque information, alongside a coin to indicate scale.**

A replaceable sphere-tipped Coordinate Measuring Machine (CMM) stylus is attached to the front face of the force sensor, and a handle to the rear, allowing a user to drag the probe tip over the surface being measured. The interchangability of the probe tip is important, since the curvature of the contact area kinematically filters the probe motion and thus impacts the acquired data.

As the surface is being probed, the force/torque measurements from the Nano17 are sampled at 5kHz using a 16-bit A/D converter (National Instruments, Austin, Texas, USA). The static gravitational load due to the probe tip is compensated for based on the measured orientation of the probe. The force and torque measured at the force sensor are transformed to the center of the probe tip to compute the contact force on the tip.

In addition to measuring force and torque, the probe's motion is tracked to provide simultaneous position data. The probe is tracked using a six-camera motion-capture system (Vicon Peak, Lake Forest, CA, USA). Several small retroreflective optical markers are attached to the probe, allowing the camera system to record and reconstruct the probe's position and orientation at 60Hz. The reconstructed position is accurate to less than 0.5mm.

The object being measured is also augmented with optical tracking markers, so the configuration of the probe with respect to the object is known even when the user moves the object to access different locations on the surface. The object is scanned with a Polhemus FastScan laser scanner (Polhemus, Colchester, VT, USA) to generate a mesh representation of the object's surface. The manufacturer reports an accuracy of 1mm for the surface. A water-tight triangular mesh is extracted from the scans using a fast RBF method. The location of the optical tracking markers are included in the scan to allow registration of the surface geometry with the motion capture data acquired during contact measurement. Figure 59 shows an example data series acquired with our setup. The full data set is available in the public repository (see Section 7.7).

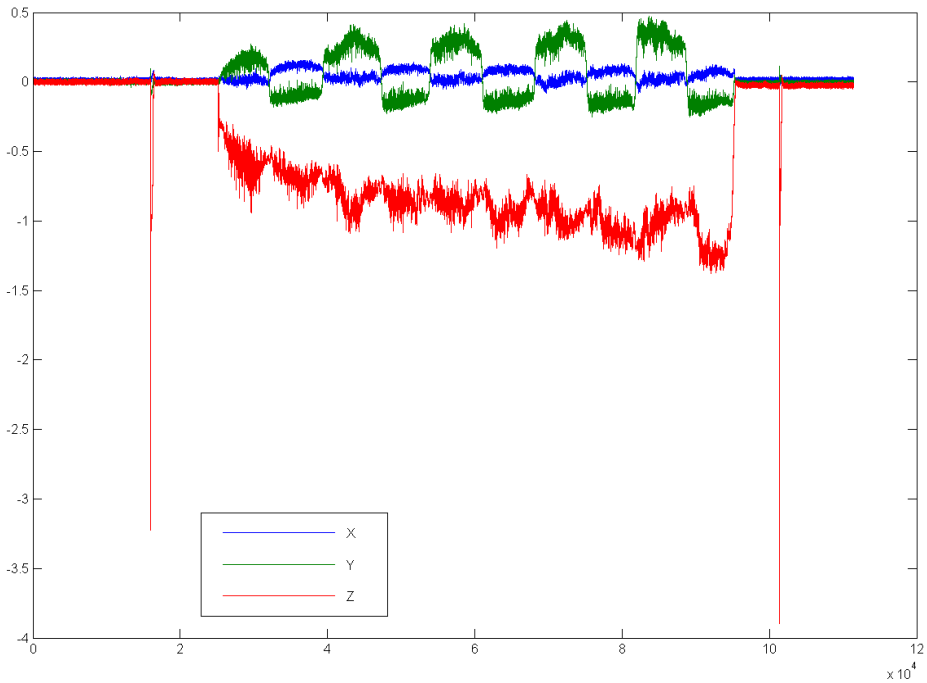



**Figure 59. Data collected from our scanning apparatus. Normal (z) forces are indicated in red, tangential (x,y) forces are indicated in green and blue. The data presented here represent a scanning motion, primarily on the y axis, on a flat plane. Brief initial and final taps were added to aid registration of force and motion data; they are visible in the normal force.**

Our initial scanning effort has focused on rigid objects, to constrain the analysis to static geometry.

## 7.3 Data Processing

Given a set of scanned trajectories, we evaluate a haptic rendering algorithm by feeding a sequence of scanned probe positions into the algorithm and comparing the computed forces to the physically-scanned forces. For penalty-based haptic rendering algorithms, this requires a pre-processing step to create a virtual trajectory that is inside the virtual representation of the scanned object.

This section will describe this process, which can be summarized in three stages:

1. Pre-processing of a scanned trajectory to allow direct comparison to rendered trajectories.
2. Computation of rendered forces and a surface contact point trajectory by the haptic rendering algorithm that is being evaluated, using the pre-processed input positions.
3. Computation of performance metrics from the output of the haptic rendering system.

Figure 60 summarizes this process.

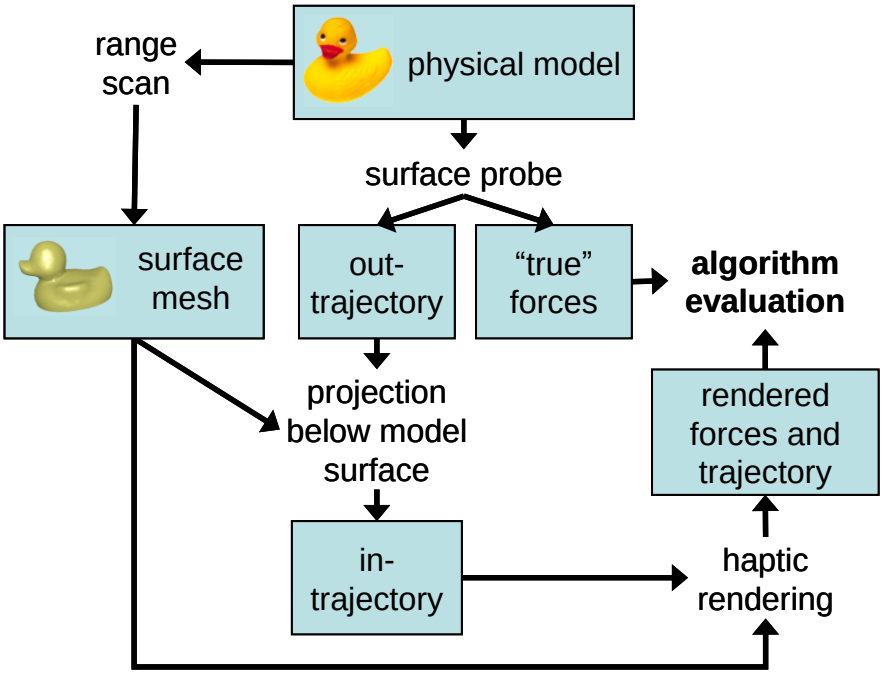



**Figure 60. An overview of our data processing and algorithm evaluation pipeline. An object is scanned, producing a 3D geometric model and an out-trajectory. An in-trajectory is synthesized from this out-trajectory and is fed as input to a haptic rendering system, which produces force and trajectory information. This information can be compared to the physically-scanned forces and the original trajectory.**

### 7.3.1 Data pre-processing

The haptic rendering algorithms on which we have performed initial analyses are penalty-based: the virtual haptic probe is allowed to penetrate the surface of a simulated object, and a force is applied to expel the haptic probe from the object. A physical (real-world) probe scanning the surface of a physical object never penetrates the surface of the object. Therefore a virtual scanning trajectory is not expected to be identical to a physical trajectory, even if a user intends to perform the same probe motions on the real and virtual objects. We therefore perform a pre-processing step that – given a physical scanning trajectory – generates a sub-surface trajectory that (under ideal conditions) produces a surface contact trajectory that is equivalent to the

scanned trajectory. This allows a direct comparison of a trajectory collected from a haptic simulation with the ideal behavior that should be expected from that simulation. We refer to an ideal trajectory (one in which the probe never penetrates the surface of the object) as an "out-trajectory", and a trajectory that allows the probe to travel inside the object as an "in-trajectory". Figure 61 demonstrates this distinction.

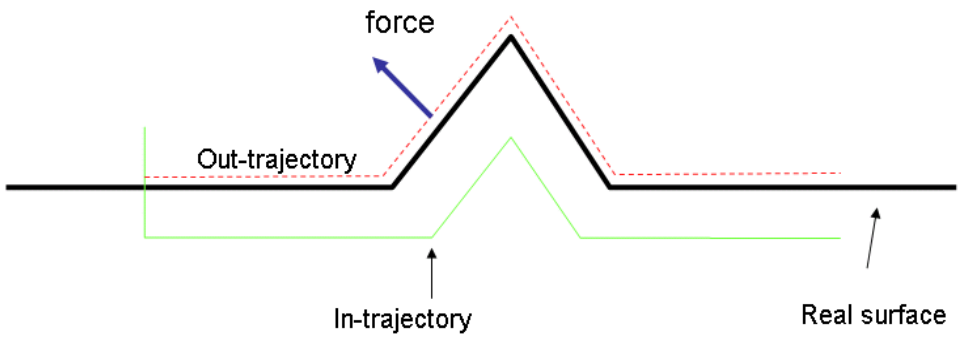



**Figure 61. An "out-trajectory" represents the path taken by a physical probe over the surface of an object; a haptic rendering algorithm typically approximates this trajectory with an "in-trajectory" that allows the probe to enter the virtual object.**

The penetration depth (the distance between the in- and out-trajectories) of a virtual haptic probe into a surface is generally dependent on an adjustable spring constant, which is an input to the algorithm and should be considered part of the system that is under evaluation; this constant is reported along with all results in our online repository. The spring constant is assumed to be homogeneous for purposes of the present analysis.

Typically, penetration depth and the resulting penalty force are related to this spring constant according to Hooke's Law:

$$\mathbf{f_p} = -k\mathbf{x} \qquad (1)$$

Here $\mathbf{f_p}$ is the penalty force vector, k is the scalar stiffness constant, and $\mathbf{x}$ is the penetration vector (the vector between the haptic probe position and a surface contact

point computed by the haptic rendering algorithm). We use this relationship to compute a corresponding in-trajectory for a physically-scanned out-trajectory.

Surface normals are computed at each point in the out-trajectory, using the scanned geometric model of the object. These surface normals are then used to extract the normal component of the recorded force at each point. Each point in the sampled out-trajectory is then converted to a corresponding point in the in-trajectory by projecting the surface point into the object along the surface normal, by a distance inversely proportional to the chosen stiffness and directly proportional to the recorded normal force (for a given normal force, higher stiffnesses should result in lower penetration depths):

$$\mathbf{p}_{in} = \mathbf{p}_{out} - \mathbf{F}_n / k \qquad (2)$$

Here $\mathbf{p}_{in}$ and $\mathbf{p}_{out}$ are corresponding in- and out-trajectory points, $\mathbf{F}_n$ is the recorded normal force at each point, and k is the selected stiffness constant. This relationship is illustrated in Figure 62. Each in-trajectory point is assigned a timestamp that is equal to the corresponding out-trajectory point's timestamp.

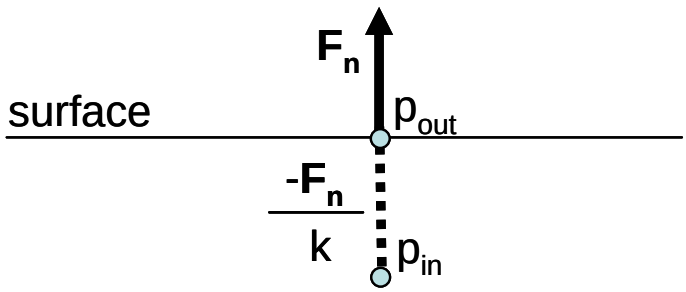



**Figure 62. Computation of an in-trajectory point from a sampled out-trajectory point.**

Following this computation, the in-trajectory corresponding to a physical out-trajectory is the path that a haptic probe would need to take in a virtual environment so that the *surface contact point* corresponding to that haptic probe path precisely follows the sampled out-trajectory.

### 7.3.2 Trajectory processing

The input to a haptic rendering algorithm is typically a geometric model of an object of interest and a series of positions obtained from a haptic interface. For the present analysis, we obtain a geometric model from the laser-scanning system described in Section 7.1, and we present a stream of positions – collected from our position-tracking system – through a "virtual haptic interface". From the perspective of a rendering algorithm implementation, this interface plays the role of a haptic device that is able to report its position in Cartesian space.

Given an in-trajectory computed from a physical out-trajectory, we can thus simulate a virtual haptic interaction with an object, which will produce a stream of forces and – in the case of many common haptic rendering algorithms – a new out-trajectory (which we refer to as a "rendered trajectory"), representing the path that a virtual contact point traveled on the surface of the virtual object.

The computational complexity of this simulation is identical to the case in which a haptic interface is used interactively, allowing assessment of computational performance in addition to algorithm output.

### 7.3.3 Metric extraction

Each time an in-trajectory is fed through a haptic rendering algorithm, producing a stream of forces and surface contact point locations, we collect the following evaluation metrics:

- *Output force error:* the difference between the forces produced by the haptic rendering algorithm and the forces collected by the force sensor. This is summarized as a root-mean-squared Euclidean distance, i.e.:

$$e = \frac{1}{N}\sum_{i=1}^{N}\left\|\vec{F}p_i - \vec{F}r_i\right\| \qquad (3)$$

Here N is the number of samples in the out-trajectory, $\mathbf{Fp_i}$ is the physically-scanned force at sample *i* and $\mathbf{Fr_i}$ is the rendered force at sample *i*. This metric is referred to as "RMS Force Error" in Section 7.4. The physically-scanned forces have been resampled to align in time with the position samples.

- *Output position error:* the difference between the surface contact point position produced by the haptic rendering algorithm and the physically sampled out-trajectory. This can also be summarized as a root-mean-squared Euclidean distance, although we have found that it is more valuable to collect the cases that exceed a threshold instantaneous error, representing "problematic" geometric cases.

- *Computational cost:* the mean, median, and maximum numbers of floating-point operations required to a compute a surface contact point and/or penalty force and the floating-point operation count for the complete trajectory. While this is not a truly platform-independent measure of computational complexity, it scales well among CPU speeds and is roughly proportional to computation time on a particular CPU.

We do not present these metrics as a comprehensive representation of haptic rendering performance, rather we present them as examples of immediately-useful data that can be extracted using our data collection system, data repository, and offline processing approach. We anticipate that future work and future contributions by the haptics community will expand the set of available metrics and assess their correlations to the perceptual quality of haptic environments.

## 7.4 Experiments and Results

We used the analyses discussed in Section 7.3 to conduct four experiments that attempt to quantify and compare haptic rendering algorithms. Specifically, we explored:

1. The relative accuracy and computational cost of a haptic proxy algorithm and a rendering scheme based on voxel sampling.
2. The impact of simulated friction on the accuracy of haptic rendering and the use of ground truth data for friction identification.
3. The impact of mesh resolution on the accuracy of haptic rendering.
4. The impact of force shading on the accuracy of haptic rendering.

For consistency, these analyses have all been performed using the same model (a scanned plastic duck) and input trajectory (see Figure 63), which is available in the online repository.

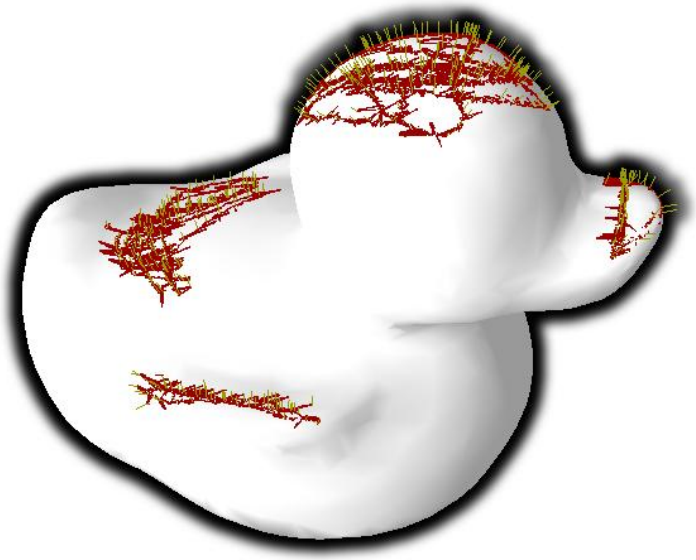


**Figure 63. The model and scanned trajectory used for the experiments presented in Section 7.4.**

These results are presented as examples of analyses that can be derived from our data sets, and their generalization to a wider variety of rendering algorithms, models, and trajectories is left for future work and is the primary goal of our online repository.

### 7.4.1 Proxy-based vs. voxel-based rendering

Our approach was used to compare the computational cost and force errors for a public-domain implementation [42] of the haptic proxy (god-object) algorithm [218] and a voxel-based rendering scheme [116], and to assess the impact of voxel resolution on rendering accuracy. This analysis does not include any cases in which the proxy provides geometric correctness that the voxel-based rendering could not; i.e. the virtual haptic probe never "pops through" the model.

Voxel-based rendering was performed by creating a fixed voxel grid and computing the nearest triangle to each voxel center. The stored triangle positions and surface normals are used to render forces for each voxel through which the probe passes.

Results for the proxy algorithm and for the voxel-based algorithm (at two resolutions) are summarized in Table 4, including the computational cost in floating-point operations, the initialization time in seconds (on a 1.5GHz Pentium), and the memory overhead. We observe that the voxel-based approach offers comparable force error and a significant reduction in floating-point computation, at the cost of significant preprocessing time and memory overhead, relative to the proxy (god-object) approach. It should be noted that analysis of this particular trajectory does not capture the fact that the proxy-based approach offers *geometric* correctness in many cases where the voxel-based approach would break down. We will discuss this further in section 7.5.

| **Algorithm** | **Voxel resolution** | **RMS force error (N)** | **Floating-point ops** | **Init time (s)** | **Memory (MB)** |
|---|---|---|---|---|---|
| voxel | $32^3$ | .136 | 484K | 0.27 | 1.0 |
| voxel | $64^3$ | .130 | 486K | 2.15 | 8.0 |
| proxy | N/A | .129 | 10.38M | 0.00 | 0.0 |

**Table 4. Accuracy and cost of haptic rendering using proxy- and voxel-based rendering schemes.**

### 7.4.2 Friction identification and evaluation

Our approach was used to evaluate the impact of simulated friction on the accuracy of haptic rendering, using a public-domain implementation [42] of the friction-cone algorithm [76]. This analysis also demonstrates the applicability of our approach for identifying rendering parameters – in this case a friction radius – from ground-truth data.

This analysis uses the friction cone algorithm available in CHAI 3D (version 1.31). The in-trajectory derived from the physical-scanned (raw) trajectory is fed to CHAI for rendering, and the resulting forces are compared to the physically-scanned forces. The coefficient of dynamic friction is iteratively adjusted until a minimum error between the physical and rendered forces is achieved. Static (stick-slip) friction was not considered for this analysis.

Results for the no-friction and optimized-friction cases are presented in Table 5, including the relative computational cost in floating-point operations. We observe that the trajectory computed with friction enabled contains significantly lower force-vector-error than the no-friction trajectory, indicating a more realistic rendering, with only a slightly higher computational cost.

| **Friction radius (mm)** | **RMS force error (N)** | **Flops** |
|---|---|---|
| 0.0000 (disabled) | 0.132 | 10.4M |
| 0.3008 | 0.067 | 10.8M |

**Table 5. Rendering accuracy with and without simulated dynamic friction.**

### 7.4.3 Impact of mesh resolution

Our approach was used to assess the impact of varying mesh resolution on the accuracy of haptic rendering. This is a potentially valuable application of our data, since mesh resolution is often varied to trade off performance for accuracy for specific applications, and the use of ground truth data will allow application developers to select minimal models that meet application-specific accuracy bounds.

The haptic proxy algorithm was provided with an in-trajectory and with eight versions of the duck model, each at a different tessellation level. The results for each resolution are presented in Table 6 and Figure 64. We observe that the error is fairly stable for a large range of resolutions between 1000 and 140000 triangles, and increases sharply for lower resolutions.

| **Model size (kTri)** | **Flops** | **RMS force error (N)** | **Relative error** |
|---|---|---|---|
| 0.2 | 9.7136M | 0.085 | 9.92 |
| 0.5 | 10.361M | 0.031 | 3.55 |
| 1 | 9.7921M | 0.031 | 3.61 |
| 3 | 10.380M | 0.022 | 2.61 |
| 6 | 10.560M | 0.022 | 2.61 |
| 9 | 10.644M | 0.015 | 1.80 |
| 64 | 10.064M | 0.013 | 1.51 |
| 140 | 9.2452M | 0.009 | 1.00 |

**Table 6. Rendering accuracy of the duck model at various mesh resolutions, computed using the proxy algorithm. "Relative error" is computed as a fraction of the error obtained using the maximum-resolution model.**

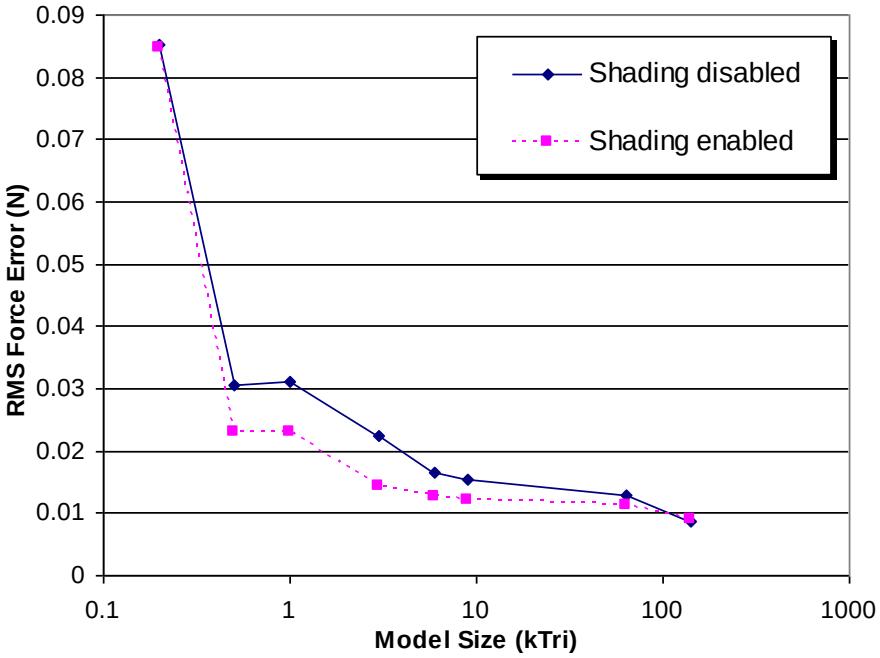



**Figure 64. Impact of mesh size (logarithmic on the x-axis) and force shading on RMS Force Error (y-axis) for our duck model, rendered with the proxy algorithm.**

### 7.4.4 Impact of force shading

The analysis presented in Section 7.4.3 was repeated with force shading [125] enabled, to quantify the impact of force shading on the accuracy of rendering this trajectory. Force shading uses interpolated surface normals to determine the direction of feedback within a surface primitive, and is the haptic equivalent of Gouraud shading.

Results are presented in Figure 64, along with the results assessing the impact of model size on rendering accuracy. We observe that for a large range of model sizes – between 1k and 10k triangles, a typical range for object sizes used in virtual environments – force shading significantly reduces the RMS force error for rendering our duck model. Note that the impact of force shading is related to the curvature of the object being rendered, and an object with smoothly-varying curvature (like our duck model) is expected to benefit significantly from force shading.

## 7.5 Discussion

We have provided a series of "ground truth" data sets for haptic rendering, acquired with a novel scanning paradigm that allows force and position data to be acquired during a natural, human-driven scanning motion. We have also presented an approach for preprocessing this data to make it suitable as input for a variety of haptic rendering algorithms, and we have provided a series of example analyses that demonstrate our approach's ability to quantitatively assess haptic rendering systems.

A key application of these data and analyses is to assess the accuracy of a particular haptic rendering system and to approximately bound the difference between the forces experienced by a user through a haptic interface and the forces the user would experience performing the same interactions with a real object. This analysis can also be used to compare haptic rendering algorithms more objectively: if one algorithm consistently produces a lower force error relative to a real data set than another algorithm, it is objectively "more realistic" by our metrics. In this context, our ground truth data set and preliminary analysis techniques may play a role in haptics similar to the role played by [169] in stereo computer vision.

This approach has an application not only in evaluating published rendering systems, but also in debugging individual implementations. Debugging haptic rendering systems is often difficult relative to debugging other computer systems, due to the hard-real-time constraints, the nondeterminism introduced by physical devices, and the difficulty of reliably replicating manual input. Our approaches and our data sets allow a developer to periodically test a haptic rendering system via a series of objective evaluations, and thus rapidly identify problems and isolate the changes that caused them.

We have also provided an objective series of input data that can be used to evaluate the *computational performance* of an algorithm. In this context, our data sets and analyses provide a "haptic benchmark", analogous to the rendering benchmarks available to the graphics community, e.g. 3DMark (Futuremark Corp). Computational performance of a haptic rendering system can vary significantly with input, but it is difficult to describe and distribute the input stream used to generate a performance

analysis result. By providing a standard data series and a set of reference results, we present a performance benchmark that authors can use to describe algorithmic performance. This is particularly relevant for objectively presenting the value of optimization strategies for rendering and collision detection whose primary value may lie in performance improvements. Performance results are still dependent on the platform used to generate the results, but this information can be reported concisely along with results.

The analyses presented here have focused primarily on "force correctness", with the ultimate metric of algorithmic correctness being the accuracy of output forces relative to ground truth forces. However, the use of standardized, pre-recorded haptic input data is also suited to assessing the *geometric* correctness of rendering algorithms, and for identifying anomalous cases that cause incorrect behavior in haptic rendering systems.

For example, Figure 65 illustrates a problematic geometry that can be captured by our analysis approach. In this case, for certain stiffness values and angles of extrusion (i.e. "bump sharpness"), the surface contact point produced by the proxy algorithm becomes "stuck" on the bump, producing an incorrect trajectory that misrepresents object geometry. Our approach allows a rapid evaluation of this geometry using a variety of synthetic models and a variety of algorithmic parameters (friction values, stiffnesses), allowing quantification of such problematic cases for particular renderer implementations. These cases are very difficult to reliably isolate when a user and physical device are in the debugging loop.

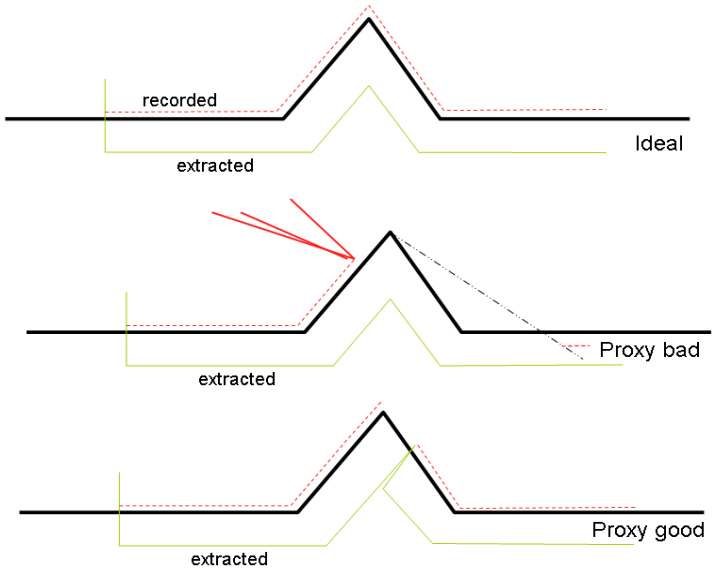



**Figure 65. This failure case for the proxy algorithm is an example of a geometric anomaly that can be captured and quantified using pre-recorded trajectories.**

Our current approach and available data sets, however, suffer from significant limitations. While a direct comparison of an algorithm's output forces to ground truth forces is expected to correlate to some degree with perceptual realism, it is not nearly a comprehensive metric. Furthermore, algorithmic performance and even results are expected to vary somewhat when collected with a user and a physical device in the loop, and no set of reference data can completely capture all possible cases that may have particular impacts on various rendering algorithms. Despite these limitations, we propose that a standard approach to haptic rendering analysis and standard data series will significantly enhance the quality and objectivity of haptic rendering system evaluation. In the following section, we will discuss future work and planned improvements to our online repository that will broaden the applicability of our data and methods.

## 7.6 Future Work

To address the limitations discussed in the previous section, future work will add both data and additional analyses to our repository. In particular, we hope to capture a wide variety of geometries, material types, contact pressures, and contact trajectories.

Subsequent acquisitions will focus on adding more complex contact shapes (our current probe approximates a single point of contact).

Furthermore, the simple RMS force error metric used in this paper is not expected to be an optimal representation of *perceptual* accuracy of haptic rendering. Future work will include the development and psychophysical evaluation of more appropriate metrics for "haptic correctness".

Given a sufficient variety of data, our approach also may have value in the automated optimization of various parameters used in haptic rendering; the identification of a dynamic friction coefficient in section 7.4.2 is a preliminary example of this application. Future work will include the generalization of this optimization scheme to a wider variety of parameters, e.g. static friction, local compliance, roughness, and haptic texture.

## 7.7 Data repository

To provide a standard reference that can be used by the community for evaluation of haptic rendering systems, the data, methods, and results discussed in this paper are publicly available at:

http://jks-folks.stanford.edu/haptic_data/

# 8 Conclusion and Future Work

## 8.1 Summary

This thesis has presented techniques for haptic rendering and physical simulation that are relevant to applications in virtual surgery. We began with a description of our simulation environment for virtual bone surgery, presenting several haptic rendering techniques and evaluating the construct validity of our simulator. We subsequently discussed several related bodies of work, focusing on problems related to – but not limited to – our simulator.

The organization of this dissertation parallels our development process: this project has proceeded in close collaboration with surgeons, and that collaboration has revealed the essential technical problems whose solutions would contribute to effective simulation. Our evaluation of haptic mentoring, though performed in an abstract psychophysical context, emerged from the observation that haptic feedback could be effective as a demonstration mechanism in a medical simulator. Our approach to preparation and calibration of deformable objects emerged as our simulator progressed toward incorporating soft tissues and required objects that were both patient-specific (and thus prepared with little manual intervention) and realistic. A number of haptic rendering and data processing techniques emerged in the development of our simulator and the other projects presented in this thesis; Section 6 presented some of those techniques in a general context to broaden their applicability. Finally, our approach to validating haptic rendering algorithms emerged from our need to confirm and improve the realism of our simulator's rendering techniques. The use

of ground truth data and data-driven methods in general form a common thread among much of the work presented here.

## 8.2 Lessons Learned

### 8.2.1 The Role of Surgeons in Simulator Development

Many simulation projects to date have focused on – or have been motivated by – specific technical problems, often problems related to soft-body deformation. This has been essential to the development of core simulation technology, and work of this nature will likely be of continued importance until accurate, interactive deformation models are widely available to simulator developers.

On the other hand, our collaborative approach to simulator design incorporates engineers, computer scientists, medical faculty, and experts in the pedagogical aspects of medical simulation. This approach will likely be essential as medical simulation matures and becomes a standard part of clinical training and surgical practice. It goes without saying that an effective simulator requires accurate content that can only be delivered in cooperation with medical personnel. But on a higher level, surgeons are, ultimately, the "customers" of medical simulation projects, and can provide invaluable advice on the level of required realism, which may vary tremendously across applications (for example, training residents may require more precision than refreshing experts) and across surgical procedures. While computer scientists can often get stuck on improving overall realism (particularly graphical realism), surgeons are able to describe the sources of feedback that are critical to surgical training, which may be visual or non-visual, and some visual subtleties may even be *enhanced* by non-photorealistic rendering. This is the approach we've taken with our simulation environment; overall realism was deemed to be less important than accurately providing the visual, haptic, and auditory cues that guide surgical decision-making. And within the set of those cues that are visual, we focused on rendering the features that surgeons identified as most relevant. For example, we determined that nonrealistic bone rendering is acceptable, but that it is critical for soft-tissue structures to become visible through drilled bone as the bone becomes thin.

Furthermore, surgeons will ultimately govern the adoption of simulation technology. Surgeons make decisions regarding resident curricula and departmental budgets, thus their enthusiasm – in addition to their intuition – is critical to progress in this area. Many simulation projects will begin and end according to the cycle of Ph.D. students in technical disciplines, but projects that last and become part of surgical curricula – and are eventually disseminated beyond the developing institutions – will be those that have garnered the support and enthusiasm of medical departments.

Finally, tuning of critical constants in simulation is often best performed with assistance from experienced surgeons. Nearly every algorithm for deformation or haptic rendering requires adjusting some set of values that affect the interactive behavior of the system, or control tradeoffs between performance and accuracy. And while these values often have a theoretical basis and can be set to "correct" values or can be set using ground truth data, it may be more efficient in some cases to allow experienced surgeons to adjust them interactively. Similarly, some systems may be subject to noise or other disturbances, such that values set based on theory or ground truth may still yield imperfect simulations. In these cases, it may also be appropriate to allow a surgeon to adjust on-screen dials or configuration files to optimize system behavior. Furthermore, it is often the case that simulations need not – or *should* not – be perfectly realistic, and in these cases the subjective impressions of target users are *better* reflections of "correctness" than theory or ground truth. For example, a preoperative rehearsal system might be best run at speeds significantly faster than real-time, to minimize rehearsal time, so a "realistic" drilling simulation may not be appropriate.

In this dissertation, we have allowed certain parameters to be calibrated manually by surgeons, where such calibration was tractable and not impractically time-consuming. For example, bone removal rates and stiffness values for our drilling simulation were bound to on-screen widgets that were adjusted over time by collaborating surgeons. On the other hand, Section 5 focuses on automatically calibrating complex deformable materials, whose parameter sets are both unintuitive and impractically large, and thus cannot be calibrated manually.

### 8.2.2 Observations on User Interfaces for Virtual Surgery

Much of the feedback that surgeons have provided about our simulation environment has focused on user interface. Surgeons are generally quite comfortable using haptic devices for drilling (Figure 66a), find the use of digital foot pedals (Figure 66b) to be sufficiently representative of the pneumatic foot pedals used intraoperatively, and report that their experience viewing operative fields through a surgical microscope translates well to viewing virtual scenes through the binocular stereo display used in our prototype (Figure 66c). However, surgeons have a particularly difficult time adjusting to switching tools and manipulating the camera in the virtual environment. This is consistent with our anecdotal observations collected during the course of the experiment presented in Section 3; we noted that surgeons had some difficulty re-creating the microscope (virtual camera) positions they are accustomed to, and thus spent a disproportionate amount of time manipulating the camera

.

**(a)** **(b)** **(c)**

**Figure 66. Components of our surgical workstation. (a) Haptic devices for drilling and suction, (b) foot pedals for enabling the drill/suction tools and zooming the camera, (c) binocular stereo display.**

Microscopes used intraoperatively for microsurgical procedures generally have six degrees of freedom, all of which are simultaneously adjusted when the surgeon releases a brake on the device. Friction and inertia limit rapid movement. Critically, the surgeon himself moves with the microscope, a phenomenon that is challenging to replicate in simulation without introducing a costly and ergonomically difficult display platform. Similar challenges arise in simulating open procedures, where a surgeon himself has six degrees of "camera" freedom and moves continuously.

We experimented with several approaches to providing six degrees of camera control in our simulator, and hope to conduct a formal study in the future that evaluates the strengths and weaknesses of each. Here we will briefly discuss some of the approaches we tried, hopefully offering some insight to those developing similar systems and choosing among camera control strategies.

We initially mapped the six degrees of freedom of a six-degree-of-freedom (three active, three passive) haptic device to the six degrees of freedom of the camera. This approach presents two significant problems. Firstly, because haptic devices with this (typical) configuration cannot be actively oriented and do not have infinite ranges of motion in any degree of freedom (rotational degrees are particularly limited), device movement must be mapped to camera movement in a relative, "clutched" sense. This is not necessarily problematic, but it does not reflect the feel of an absolutely-positioned surgical microscope. Furthermore, we found the rotational degrees of freedom of the haptic device – which is small and has limited friction/inertia – to be more sensitive than the corresponding degrees of freedom found on a microscope. Consequently, it was difficult to translate the virtual camera without simultaneously rotating it.

We solved the latter problem by discarding the rotational degrees of freedom of the haptic device when manipulating the camera, and mapping the three translational degrees of freedom of the haptic device to either the rotational *or* translational degrees of freedom of the camera, depending on the state of a button. When one button is pressed, translating the device translates the camera; when a different button is pressed, translating the device rotates the camera.

Once an intuition is developed for this scheme, we find that it is much easier to control than the scheme employing all six degrees of device freedom, since only three degrees of freedom are active at one time. However, building such an intuition for this control scheme is difficult for many users. Mapping device translation to camera rotation is fairly straightforward; we explain it to novices as grabbing the back of a video camera that is rotating on a tripod. However, users have significant trouble *planning* camera movements, which requires decomposing a desired camera state

(position and orientation) into the translational and rotational paths necessary to arrive at that state. A user can often imagine himself at a particular place in the scene, looking in a particular direction, but it is often unclear what sequence of rotations and translations would move the camera from the current state to the imagined or desired state.

Future work will explore the psychophysical aspects of this problem in more detail, building on existing literature in the virtual reality community on first-person navigation and camera/scene manipulation ([72], [190], [192], [47], [216], [48], [208], [14], [46], [188], [187], [158], [29]). We are also experimenting with isometric, six-degree-of-freedom camera controllers based on optical tracking and/or strain gauges, and with absolutely-positioned camera controllers based on rigid linkages with potentiometers or encoders.

## 8.3 Future Work

The individual sections of this dissertation discussed future work related specifically to each of the topics discussed. This section will first discuss work that will enable the integration of the components presented in this dissertation, then will propose several broader areas for research in surgical simulation.

### 8.3.1 Integration into the Bone Surgery Simulator

Future work at Stanford on the simulation environment for bone surgery will include the integration of the components discussed here, particularly in applying the work developed in Sections 4, Section 5, and Section 7 to the environment discussed in Section 3. Haptic mentoring – discussed in an abstract context in Section 4 – offers tremendous possibilities for surgical simulation, and a basic implementation of this technique was discussed in Section 3. Before this can be more effectively integrated into a surgical trainer, future work will include selecting exactly the haptic features that should be presented to a trainee, evaluating the tradeoff between position gain and force playback, and formally evaluating the efficacy of this approach in teaching surgical skills. The techniques presented in Section 5 prepare anatomy models for

real-time deformation, and their incorporation into our simulation environment – for example to represent tumors, muscle tissue, etc. – will require further work on image segmentation (to construct surface models from image data), collision detection, and haptic rendering (the simulation techniques referred to in Section 5 do not proceed at haptic rates and do not trivially yield haptic feedback forces).

### 8.3.2 New Directions

The remainder of this section proposes several broader areas for research in surgical simulation.

#### 8.3.2.1 Patient-specific Simulation

A key application of medical simulation will likely be patient-specific simulation, i.e. practicing a particular procedure on an actual patient's anatomy, pathology, and physiology. This will allow surgeons to rehearse preoperatively in cases where a surgery may be challenging, where adverse events are considered likely, or where multiple approaches may be possible. Furthermore, patient-specific simulation will allow residents to review actual cases before or after procedures to obtain a deeper understanding of intraoperative events. However, several challenges need to be addressed before patient-specific simulation becomes a reality.

The first and most critical is automated segmentation of medical images. It is expected that a simulation environment will treat different tissue types differently (for example, bone might be voxelized as per Section 3 of this dissertation, while a muscle might be prepared as a deformable object as per Section 5). Segmentation, however, is still a largely open problem (reviews of core techniques are provided in [149], [39], [115], [202], [22]). Achieving success in patient-specific simulation will thus require domain-specific identifications of which tissue types need to be extracted per-patient, which can be used from canonical atlases, which can be segmented with limited accuracy, etc. A closely-coupled problem is *image registration*: the alignment of multiple images – captured at different times or via different modalities – to supplement the information available in a single image. Automatic registration is also considered an open problem and has yet to emerge as a common feature in

commercial image-processing platforms. [109] provides a review of core techniques in this area. Registration is particularly relevant for patient-specific simulation, as certain critical structures may be visible in some modalities but not in others, so multiple images are required to gather all the information necessary for a complete rehearsal. For example, CT images of the temporal bone region simulated by our environment generally present crisp images of bone boundaries, while MR images reveal nerves and blood vessels not visible via CT.

Beyond segmentation and registration, patient-specific simulation will likely also require information about tissue properties, physiology, and pathology that are currently beyond the scope of automatic determination from image data alone. Future work will be required to assess whether using canonical values for mechanical tissue properties is sufficient for patient-specific rehearsal, or whether such properties can be automatically determined from image density values. The work on tissue property acquisition discussed in Section 2.4 will need to continue into a clinical setting, and will likely be coupled with non-invasive measurements of human tissue properties using optical imaging or video sequences.

Perhaps the most significant problem in the area of patient-specific simulation is the identification of the ideal feature set for a patient-specific simulator. In other words, what type of simulation will surgeons actually use in practice? A fully realistic simulator is unlikely to be adopted, as surgeons are unlikely to routinely spend five non-clinical hours preparing for a five-hour procedure. In general, surgeons are likely to be interested in specific components of a procedure, and providing a user interface that allows surgeons to identify and "fast-forward" to those components – perhaps based on sequences of events recorded from canonical procedures – is a target for future research. The "right" application – the tool a surgeon would want on his or her desk for frequent use – will likely sit somewhere between simulation and interactive visualization.

#### 8.3.2.2 Curriculum Development

Even given an ideal physical simulator and user interface, the successful application of medical simulation to resident training will require significant curriculum

development, integrating input from surgeons, educators, and simulation engineers. Whether this will require task-specific manual preparation for new disciplines and/or new cases is not yet clear; future work will focus both on specific curriculum development and on techniques for accelerating curriculum development, including automatic preparation of case data, improved user interfaces for interactive atlases, and design guidelines for preparing new simulation-based material.

### 8.3.3.3 Automated Evaluation

Closely related to curriculum development is the topic of automated evaluation. Simulators provide a potential mechanism for standardizing the evaluation of residents, which is currently somewhat subjective in most disciplines and varies among instructors and among institutions. Furthermore, simulators offer the possibility of immediate, trainee-specific feedback that can optimize a resident's training program and focus each user on specific problem areas.

However, significant work needs to be done to enable automated assessment of correctness and automated scoring. This may take the form of recognizing physiologically-threatening events, may depend on explicitly-defined "correct techniques" reported by surgeons, and/or may utilize comparisons to canonical procedures performed by skilled surgeons. We are currently using the data collected for the validation experiment presented in Section 3 (recorded simulated procedures, instructor-assigned scores, and reported experience levels) to build automated classifiers that separate novices from experts ([179], [178], [177]). Later we plan to extend this to automated scoring, and ultimately to an interactive training environment for temporal bone procedures. Other work in this area focuses on defining Markov models for correct surgical procedures ([163], [164], [162], [52], [86]).

### 8.3.3.4 Non-Traditional Applications

The final area for future research in medical applications that we'll highlight here is the application of simulation technology to non-traditional simulation settings, i.e. applications outside of the standard surgical training/rehearsal model.

One particularly appealing model is simulation for veterinary care, focusing on surgeries for uncommon species on which a veterinarian may be required to perform a procedure with little background. While most veterinarian will encounter primarily domestic animals (pets and livestock), a veterinarian working at a zoo, wildlife sanctuary, wildlife rehabilitation center, etc. is likely to encounter species that his or her veterinary training did not cover. Simulation – even based on a small number of canonical models for each species – offers veterinarians a mechanism for rapid practice on representative anatomy, physiology, and pathology. Future work in this area will focus largely on acquiring image data, physiology data, and tissue property information for a variety of species, and extending existing simulation work (for human applications) to take advantage of this data.

Another potential application for simulation is in training non-surgical personnel who may have to perform basic surgical procedures in crisis situations or in situations where medical personnel or hospital transport are unavailable. This may include first responders (particularly EMT's and paramedics), non-surgeon physicians, nurses, physician assistants, and military personnel. Similarly, simulation offers a mechanism for preparing even experienced surgeons for unusual surgical conditions, e.g. battlefield surgery, on-site surgery during crises, surgery in poorly-equipped hospitals, etc. Work in this area will focus largely on curriculum development, and may place only limited requirements – relative to routine resident training – on simulation precision.

Finally, simulation – likely coarse simulation – may offer an interactive mechanism for patients and their families to quickly understand procedures preoperatively, in significantly more detail than a surgeon can provide verbally. While this may not be desirable for all patients, the medical community strives to make relevant information available to patients, and simulation-based presentations may be efficient and effective in introducing surgical concepts to patients. Again, precision is unlikely to be a limiting factor here; future work in this area will focus primarily on curriculum development and user interface.

# References


[1] Abaqus/Standard and Abaqus/CAE. Abaqus, Inc., Providence, RI. http://www.hks.com/

[2] Abbott J, Marayong P, Okamura A. Haptic Virtual Fixtures for Robot-Assisted Manipulation. *12th International Symposium of Robotics Research* (ISRR), October 2005.

[3] Adams RJ, Klowden D, Hannaford B. Virtual Training for a Manual Assembly Task. *Haptics-e*, Vol. 2, No. 2, 2001.

[4] Agus M, Brelstaff G, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A, Picasso B, Franceschini SS. Physics-based burr haptic simulation: tuning and evaluation. *12th IEEE International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. Pages 128-135. IEEE Computer Society Press, April 2004.

[5] Agus M, Giachetti A, Gobbetti E, Picasso B, Franceschini SS, Zanetti G, Zorcolo A. A haptic model of a bone-cutting burr. *Proceedings of Medicine Meets Virtual Reality 2003*. Pages 4-10, IOS, Amsterdam, The Netherlands, January 2003.

[6] Agus M, Giachetti A, Gobbetti E, Zanetti G, John NW, Stone RJ. Mastoidectomy Simulation with Combined Visual and Haptic Feedback. *Proceedings of Medicine Meets Virtual Reality 2002*, pages 17-23, IOS, Amsterdam, The Netherlands, January 2002.

[7] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A. A multiprocessor decoupled system for the simulation of temporal bone surgery. *Computing and Visualization in Science*. 2002 5(1):35-43.

[8] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A. Adaptive techniques for real-time haptic and visual simulation of bone dissection. *Proceedings of the IEEE Virtual Reality Conference*, pages 102-109, IEEE Computer Society Press, March 2003.

[9] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A. Real-time Haptic and Visual Simulation of Bone Dissection. *Presence: Teleoperators and Virtual Environments*, 12(1): 110-122, February 2003.

[10] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A. Real-time Haptic and Visual Simulation of Bone Dissection. *Proceedings of the IEEE Virtual Reality Conference*, pages 209-216, IEEE Computer Society Press, February 2002.

[11] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A. Hardware-Accelerated Dynamic Volume Rendering for Real-Time Surgical Simulation. *Workshop on Virtual Reality Interactions and Physical Simulations* (VRIPHYS) 2004, September 20-21, 2004.

[12] Ahmad A, Alnoah Z, Kochman ML, Krevsky B, Peikin SR, Mercogliano G, Bailey M, Boynton R, AND Reynolds JC. Endoscopic Simulator Enhances Training of Colonoscopy in a Randomized, Prospective, Blinded Trial. *Gastrointestinal Endoscopy*, April 2003, 57(5): S1499.

[13] Antani S, Xu X, Long L, Thomas G. Partial Shape Matching for CBIR of Spine X-ray Images. *Proceedings of the 16th SPIE Symposium on Electronic Imaging*, San Jose, CA, 2004.

[14] Balakrishnan B and Kurtenbach B. Exploring Bimanual Camera Control and Object Manipulation in 3D Graphics Interfaces. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI) 1999*.

[15] Balaniuk R and Salisbury JK. Dynamic Simulation of Deformable Objects Using the Long Elements Method. *IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems 2002*: 58-65.

[16] Balaniuk R and Salisbury JK. Soft-Tissue Simulation Using the Radial Elements Method. *Surgery Simulation and Soft Tissue Modeling: International Symposium*, IS4TM 2003 Juan-Les-Pins, France, June 12-13, 2003.

[17] Bartz D and Guvit O. Haptic Navigation in Volumetric Datasets. *Second PHANToM Users Research Symposium*, Zurich, Switzerland, 2000.

[18] Basdogan C, Ho C, Srinivasan MA. Virtual Environments for Medical Training: Graphical and Haptic Simulation of Laparoscopic Common Bile Duct Exploration. *IEEE/ASME Transactions on Mechatronics*, Vol. 6, No. 3, 2001.

[19] Bentley JL. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18, 9 (Sep. 1975), 509-517.

[20] Berkley J, Turkiyyah G, Berg D, Ganter M, Weghorst S. Real-Time Finite Element Modeling for Surgery Simulation: An Application to Virtual Suturing. *IEEE Transactions on Visualization and Computer Graphics*, 10(3), 1-12.

[21] Berti G, Fingberg J, Schmidt JG. An Interactive Planning and Simulation Tool for Maxillo-Facial Surgery. *Proceedings of Medical Robotics, Navigation, and Visualization (MRNV) 2004*.

[22] Bezdek JC, Hall LO, Clarke LP. Review of MRI Image Segmentation Techniques Using Pattern Recognition. *Medical Physics*, vol. 20, no. 4, pp. 1033-1048.

[23] Bhat K, Twigg C, Hodgins J, Khosla P, Popovic Z, Seitz S. Estimating cloth simulation parameters from video. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (SCA) 2003.

[24] Bianchi G, Harders M, Székely G. Mesh Topology Identification for Mass-Spring Models. *Proceedings of Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2003, Montreal, Canada, November 2003.

[25] Bianchi G, Solenthaler B, Székely G, Harders M. Simultaneous Topology and Stiffness Identification for Mass-Spring Models based on FEM Reference Deformations. *Proceedings of Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2004. St-Malo, France, November 2004.

[26] Blevins NH, Jackler RK, Gralapp C. *Temporal Bone Dissector*. Mosby, January 1998.

[27] Bourguignon D and Cani M-P. Controlling Anisotropy in Mass-Spring Systems. *Eurographics Workshop on Computer Animation and Simulation* (EGCAS), pages 113-123, August 2000.

[28] Bouvier DJ. Double-Time Cubes: A Fast 3D Surface Construction Algorithm for Volume Visualization. *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, June 1997.

[29] Bowman D, Koller D, Hodges L. Travel in Immersive Virtual Environments: an Evaluation of Viewpoint Motion Control Techniques. *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS) 1997*.

[30] Brelstaff G, Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A, Picasso B, Franceschini SS. Towards a psychophysical evaluation of a surgical simulator for bone-burring. *Proceedings of the Second Symposium on Applied Perception in Graphics and Visualization*. Pages 139-143. ACM Press, August 2005.

[31] Bridson R, Teran J, Molino N, Fedkiw R. Adaptive Physics Based Tetrahedral Mesh Generation Using Level Sets. *Engineering with Computers* 21, 2-18 (2005).

[32] Brouwer I, Ustin J, Bentley L, Sherman A, Dhruv N, Tendick F. Measuring In Vivo Animal Soft Tissue Properties for Haptic Modeling in Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2001*.

[33] Brown J, Montgomery K, Latombe JC, Stephanides M. A Microsurgery Simulation System. *Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2001, Utrecht, The Netherlands, October 14-17, 2001.

[34] Bryan J, Stredney D, Wiet G, Sessanna D. Virtual Temporal Bone Dissection: A Case Study. *Proceedings of IEEE Visualization 2001*, 497-500, October 2001.

[35] Bumm K, Wurm J, Rachinger J, Dannenmann T, Bohr C, Fahlbusch R, Iro H, Nimsky C. An automated robotic approach with redundant navigation for minimal invasive extended transsphenoidal skull base surgery. *Minimally Invasive Neurosurgery*, 2005 Jun;48(3):159-64.

[36] CardioSkills, http://www.cardioskills.com/

[37] Cavusoglu MC, Goktekin TG, Tendick F, Sastry SS. GiPSi: An open source/open architecture software development framework for surgical simulation. *Proceedings of MMVR*, 2004:46-48.

[38] Choi K-S, Sun H, Heng P-A, Zou J. Deformable simulation using force propagation model with finite element optimization. *Computers & Graphics* 28(4): 559-568 (2004).

[39] Clarke LP, Velthuizen RP, Camacho MA, Heine JJ, Vaidyanathan M, Hall LO, Thatcher RW, Silbiger ML. MRI segmentation: methods and applications. *Magnetic Resonance Imaging*. 1995;13(3):343-68.

[40] Cohen JD, Lin MC, Manocha D, Ponamgi M. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments.

*Proceedings of the ACM Symposium on Interactive 3D Graphics*, pp. 189-196, 1995.

[41] Colgate JE and Brown JM. Factors Affecting the Z-Width of a Haptic Display. *Proceedings of the IEEE Conference on Robotics and Automation*, San Diego, CA, USA, May 1994.

[42] Conti F, Barbagli F, Morris D, Sewell C. CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes. Demo paper presented at the *Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (WorldHaptics). Pisa, Italy, 2005

[43] Cotin S, Dawson SL, Meglan D, Shaffer DW, Farrell MA, Bardsley RS, Morgan FM, Nagano T, Nikom J, Sherman P, Walterman MT, Wendlandt J. ICTS: An Interventional Cardiology Training System. *Proceedings of Medicine Meets Virtual Reality*, pages 59-65, January 2000.

[44] Cotin S, Delingette H, Ayache N. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62-73, January-March 1999.

[45] Cotin S, Shaffer D, Meglan D, Ottensmeyer M, Berry P, Dawson S. CAML: A general framework for the development of medical simulation systems. *Proceedings of SPIE* 4037:294-300.

[46] De Boeck J and Coninx K. Haptic Camera Manipulation: Extending the "Camera in Hand Metaphor". *Proceedings of EuroHaptics 2002*.

[47] De Boeck J, Cuppens E, De Weyer T, Raymaekers C, Coninx K. Multisensory Interaction Metaphors with Haptics and Proprioception in Virtual Environments. *Proceedings of NordCHI 2004*.

[48] De Boeck J, Raymaekers C, Coninx K. Expanding the Haptic Experience by Using the PHANToM Device as a Camera Metaphor. *Proceedings of the Sixth Annual Meeting of the Phantom Users Group*, 2001.

[49] De S, Kim J, Manivannan M, Srinivasan MA, Rattner D. Multimodal Simulation of Laparoscopic Heller Myotomy Using a Meshless Technique. *Proceedings of Medicine Meets Virtual Reality (MMVR)*, pages 127-132, Newport Beach, January 2002.

[50] De S, Kim J, Srinivasan MA. A Meshless Numerical Technique for Physically Based Real Time Medical Simulations. *Proceedings of Medicine Meets Virtual Reality (MMVR)*, pages 113-118, Newport Beach, January 2001.

[51] Deussen O, Kobbelt L, Tücke P. Using Simulated Annealing to Obtain Good Nodal Approximations of Deformable Bodies. *Proc. Sixth Eurographics Workshop on Simulation and Animation*, September 1995, Maastricht.

[52] Dosis A, Bello F, Gillies D, Undre S, Aggarwal R, Darzi A. Laparoscopic task recognition using Hidden Markov Models. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2005*, pp. 115-122.

[53] Duffy AJ, Hogle NJ, McCarthy H, Lew JI, Egan A, Christos P, Fowler DL. Construct validity for the LAPSIM laparoscopic surgical simulator. *Surgical Endoscscopy*, 2005 Mar;19(3):401-5.

[54] Engum SA, Jeffries P, Fisher L. Intravenous catheter training system: Computer-based education versus traditional learning methods. *American Journal of Surgery*, July 2003, Vol. 186, No. 1.

[55] Etzmuss O, Gross J, Strasser W. Deriving a Particle System from Continuum Mechanics for the Animation of Deformable Objects. *IEEE Transactions on Visualization and Computer Graphics*, v.9 n.4, p.538-550, October 2003.

[56] Everett P, Seldin EB, Troulis M, Kaban LB, Kikinis R. A 3-D System for Planning and Simulating Minimally-Invasive Distraction Osteogenesis of the Facial Skeleton. *Proceedings of Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2000*: 1029-1039.

[57] Falk V, Mourgues F, Vieville T, Jacobs S, Holzhey D, Walther T, Mohr FW, Coste-Maniere E. Augmented reality for intraoperative guidance in endoscopic coronary artery bypass grafting. *Surgical Technology International*, 2005;14:231-5.

[58] Feygin D, Keehner M, Tendick F. Haptic Guidance: Experimental Evaluation of a Haptic Training Method for a Perceptual Motor Skill. *Proceedings of the 10th IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2002.

[59] Fisher S and Lin M. Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Fields. *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS) 2001.

[60] Garroway D and Hayward V. A Haptic Interface for Editing Space Trajectories. Poster presented at *ACM SIGGRAPH & EuroGraphics Symposium on Computer Animation*. August 2004.

[61] Georgii J, Echtler F, Westermann R. Interactive Simulation of Deformable Bodies on GPUs. *SimVis 2005*: 247-258.

[62] Gettman MT, Blute ML. Critical comparison of laparoscopic, robotic, and open radical prostatectomy: techniques, outcomes, and cost. *Current Urology Reports*, 2006 May;7(3):193-9.

[63] Giachetti A, Agus M, Gobbetti E, Zanetti G. Blood, dust and mud for a temporal bone surgical simulator. *Proceedings of the 5th Conference of the European Society of Mathematical and Theoretical Biology*, July 2002.

[64] Gibson SF and Mirtich B. A survey of deformable models in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA, November 1997.

[65] Gibson SF, Samosky J, Mor A, Fyock C, Grimson W, Kanade T, Kikinis R, Lauer H, McKenzie N, Nakajima S, Ohkami T, Osborne R, Sawada A. Simulating arthroscopic knee surgery using volumetric object representations, real-time volume rendering and haptic feedback. *Proceedings of the First Joint Conference on Computer Vision, Virtual Reality and Robotics in Medicine and Medial Robotics and Computer-Assisted Surgery*, p.369-378, March 19-22, 1997.

[66] Gillespie RB, O'Modhrain M, Tang P, Zaretzky D, Pham C. The Virtual Teacher. *Proceedings of the 1998 ASME International Mechanical Engineering Congress and Exposition*, Anaheim, CA, USA, 1998.

[67] Gorman PJ, Meier AH, Rawn C, Krummel TM. The future of medical education is no longer blood and guts, it is bits and bytes. *American Journal of Surgery*. 2000 Nov. 180(5):353-356.

[68] Grantcharov TP, Carstensen L, Schulze S. Objective assessment of gastrointestinal endoscopy skills using a virtual reality simulator. *Journal of the Society of Laparoendoscopic Surgeons*. 2005 Apr-Jun;9(2):130-3.

[69] Grantcharov TP, Kristiansen VB, Bendix J, Bardram L, Rosenberg J, Funch-Jensen P: Randomized clinical trial of virtual reality simulation for laparoscopic skills training. *British Journal of Surgery* 2004, 91: 146-150.

[70] Gretarsdottir UO, Barbagli F, Salisbury JK. Phantom-X. *EuroHaptics 2003*, Dublin, Ireland.

[71] Guerraz A, Loscos C, Widenfeld HR. How to use physical parameters coming from the haptic device itself to enhance the evaluation of haptic benefits in user interface. *EuroHaptics 2003*, Dublin, Ireland.

[72] Hachet M, Reuter P, Guitton P. Camera Viewpoint Control with the Interaction Table. *Proceedings of Virtual Reality International Conference 2003*.

[73] Haluck RS, Gallagher AG, Satava RM, Webster R, Bass TL, Miller CA. Reliability and validity of Endotower, a virtual reality trainer for angled

endoscope navigation. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2002*.

[74] Haluck RS, Marshall RK, Krummel TM, Melkonian MG. Are surgery training programs ready for virtual reality? *Journal of the American College of Surgeons*. 2001 Dec. 193(6):660-665.

[75] Hariri S, Rawn C, Srivastava S, Youngblood P, Ladd A. Evaluation of a surgical simulator for learning clinical anatomy. *Medical Education*, Volume 38, Issue 8, Page 896 - August 2004.

[76] Harwin WS and Melder N. Improved Haptic Rendering for Multi-Finger Manipulation using Friction Cone based God-Objects. *Proceedings of EuroHaptics 2002*, Edinburgh, UK.

[77] HavokFX, Havok, Inc., http://www.havok.com/

[78] Hayward V and Astley OR. Performance measures for haptic interfaces. *Proceedings of Robotics Research: 7th International Symposium*, 1996.

[79] Hohne KH, Bomans M, Riemer M, Schubert R, Tiede U, Lierse W. A 3D anatomical atlas based on a volume model. *IEEE Visualization 1992*, 12 (1992) 72-78.

[80] Ho-Le K. Finite element mesh generation methods: a review and classification. *Computer Aided Design*, Vol 20(1), 27-38, 1988.

[81] Holzman RS, Cooper JB, Gaba DM. Anesthesia crisis resource management: real-life simulation training in operating room crises. *Journal of Clinical Anesthesiology*, 1995 7:675–687.

[82] http://graphics.stanford.edu/data/3Dscanrep/

[83] http://tetgen.berlios.de/fformats.examples.html

[84] http://www.osc.edu/research/Biomed/vtbone/

[85] Hu T, Tholey G, Desai JP, and Castellanos AE. Evaluation of a laparoscopic grasper with force feedback. *Surgical Endoscopy*, 2004 May;18(5):863-7.

[86] Huang J, Payandeh S, Doris P, Hajshirmohammadi I. Fuzzy classification: towards evaluating performance on a surgical simulator. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2005*, pp. 194-200.

[87] Immersion Medical, http://www.immersion.com/medical/

[88] Ingber L. Adaptive simulated annealing (ASA): lessons learned. *Control and Cybernetics*, 25, 33–54, 1995.

[89] Ingber, L. ASA-26.14. 2006. http://www.ingber.com/#ASA

[90] James D and Pai D. ARTDEFO: Accurate real time deformable objects. *Proceedings of ACM SIGGRAPH 1999*, 65–72.

[91] James D and Twigg CD. Skinning Mesh Animations. *ACM Transactions on Graphics* (*SIGGRAPH 2005*), Vol. 24, No. 3, August 2005.

[92] Jones MG, Bokinsky A, Tretter T, Negishi A. A Comparison of Learning with Haptic and Visual Modalities. *Haptics-e*, Vol. 3, No. 6, 2005.

[93] Judkins TN, Oleynikov D, Narazaki K, Stergiou N. Robotic surgery and training: electromyographic correlates of robotic laparoscopic training. *Surgical Endoscopy*, 2006, May;20(5):824-9.

[94] Kazi A. Operator performance in surgical telemanipulation. *Presence*, vol. 10, pp. 495-510, 2001.

[95] Keeve E, Girod S, Kikinis R, Girod B. Deformable modeling of facial tissue for craniofacial surgery simulation. *Computer Aided Surgery*, 1998, 3:228–38.

[96] Keeve E, Pfeifle P, Girod B, Girod S. Anatomy Based Facial Tissue Modeling Using the Finite Element Method. *Proceedings of the IEEE Conference on Visualization*, 1996, pp 21-28.

[97] Kerdok AE, Cotin SM, Ottensmeyer MP, Galea AM, Howe RD, Dawson SL. Truth cube: establishing physical standards for soft tissue simulation. *Medical Image Analysis*. 2003 Sep;7(3):283-91.

[98] Kim L, Sukhatme G, Desbrun M. A haptic rendering technique based on hybrid surface representation. *IEEE Computer Graphics and applications*, March 2004.

[99] Kirkpatrick AE and Douglas SA. Application-based Evaluation of Haptic Interfaces. *10th IEEE Haptics Symposium*, 2002, Orlando, USA.

[100] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by Simulated Annealing. *Science*, Vol 220, Number 4598, pages 671-680, 1983.

[101] Koch RM, Roth SHM, Gross MH, Zimmermann AP, Sailer HF. A Framework for Facial Surgery Simulation. *Proceedings of the 18th Spring Conference on Computer Graphics*, p33–42, 2002.

[102] Korb W, Kornfeld M, Birkfellner W, Boesecke R, Figl M, Fuerst M, Kettenbach J, Vogler A, Hassfeld S, Kornreif G. Risk analysis and safety assessment in surgical robotics: A case study on a biopsy robot. *Minimally Invasive Therapy & Allied Technologies*, 2005;14(1):23-31.

[103] Kry PG, James DL, Pai DK. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. *Proceedings of ACM SIGGRAPH 2002 Symposium on Computer Animation*.

[104] Lawrence DA, Pao LY, Dougherty AM, Salada MA, and Pavlou Y. Rate-Hardness: a New Performance Metric for Haptic Interfaces. *IEEE Transactions on Robotics and Automation*, 16(4): 357-371, Aug. 2000.

[105] Lindblad AJ, Turkiyyah GM, Weghorst SJ, Berg D. Real-Time Finite Element Based Virtual Tissue Cutting. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2006*, 24-27 January 2006, Long Beach, CA.

[106] Litofsky NS, Bauer AM, Kasper RS, Sullivan CM, Dabbous OH. Image-guided resection of high-grade glioma: patient selection factors and outcome. *Neurosurgical Focus*, 2006, Apr 15;20(4):E16.

[107] Lorensen WE and Cline HE. Marching Cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH* 1987, 21:163–169.

[108] Luring C, Bathis H, Tingart M, Perlick L, Grifka J. Computer assistance in total knee replacement - a critical assessment of current health care technology. *Computer Aided Surgery*, 2006, Mar;11(2):77-80.

[109] Maintz JB and Viergever MA. A survey of medical image registration. *Medical Image Analysis*. 1998 Mar;2(1):1-36.

[110] Massie TH and Salisbury JK. The PHANTOM Haptic Interface: A Device for Probing Virtual Objects. *Symposium on Haptic Interfaces for Virtual Environments*. Chicago, IL, Nov. 1994.

[111] Matlab 7.0 (R14). The MathWorks, Inc., Natick, MA. http://www.mathworks.com/

[112] Mauch S. Closest Point Transform (open-source software): http://www.acm.caltech.edu/~seanm/projects/cpt/cpt.html

[113] Mauch S. Efficient Algorithms for Solving Static Hamilton-Jacobi Equations. PhD thesis, 2003.

[114] McDougall EM, Corica FA, Boker JR, Sala LG, Stoliar G, Borin JF, Chu FT, Clayman RV. Construct validity testing of a laparoscopic surgical simulator. *Journal of the American College of Surgeons*. 2006 May;202(5):779-87.

[115] McInerney T and Terzopoulos D. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, vol. 1, no. 2, 1996/1997, 91-108.

[116] McNeely WA, Puterbaugh KD, and Troy JJ. Six degree-of-freedom haptic rendering using voxel sampling. *Proceedings of ACM SIGGRAPH 1999*, pages 401-408.

[117] McNeely WA, Puterbaugh KD, Troy JJ. Voxel-Based 6-DOF Haptic Rendering Improvements. *Haptics-e*, vol. 3, 2006.

[118] Meehan M, Morris D, Maurer C, Antony A, Barbagli F, Salisbury K, and Girod S. Virtual 3D Planning and Guidance of Mandibular Distraction Osteogenesis. Computer Aided Surgery, Volume 11, Number 2, p51-62, March 2006.

[119] Meglan DA, Raju R, Merril GL, Merril JR, Nguyen BH, Swamy SN, Higgins GA. The Teleos virtual environment toolkit for simulation-based surgical education. *Proceedings of MMVR*, 1996:346-51.

[120] Melvin WS, Dundon JM, Talamini M, Horgan S. Computer-enhanced robotic telesurgery minimizes esophageal perforation during Heller myotomy. *Surgery*, 2005 Oct;138(4):553-8; discussion 558-9.

[121] Mendoza C and Laugier C. Simulating soft tissue cutting using finite element models. *Proceedings of the International Conference on Robotics and Automation (ICRA) 2003*: 1109-1114.

[122] Mohr CJ, Nadzam GS, Curet MJ. Totally robotic Roux-en-Y gastric bypass. *Archives of Surgery*, 2005 Aug;140(8):779-86.

[123] Montgomery K, Bruyns C, Wildermuth S, Heinrichs L, Hasser C, Ozenne S, Bailey D. Surgical Simulator for Operative Hysteroscopy. *IEEE Visualization 2001*, San Diego, California, October 14-17, 2001.

[124] Montgomery K, Heinrichs L, Bruyns C, Wildermuth S, Hasser C, Ozenne S, and Bailey D. Surgical Simulator for Hysteroscopy: A Case Study of Visualization in Surgical Training. *12th IEEE Visualization Conference*, San Diego, CA, 2001, pp. 449-452.

[125] Morgenbesser HB and Srinivasan MA. Force shading for haptic shape perception. *Proceedings of ASME Dynamic Systems and Control Division*, (DSC-Vol.58): 407-412, 1996.

[126] Morris D, Girod S, Barbagli F, Salisbury K. An Interactive Simulation Environment for Craniofacial Surgical Procedures. *Proceedings of MMVR (Medicine Meets Virtual Reality) XIII*, Long Beach, CA. Studies in Health Technology and Informatics, Volume 111, 2005.

[127] Morris D, Sewell C, Barbagli F, Blevins N, Girod S, and Salisbury K. Visuohaptic Simulation of Bone Surgery for Training and Evaluation. To appear in *IEEE Computer Graphics and Applications*, November 2006.

[128] Morris D, Sewell C, Blevins N, Barbagli F, and Salisbury K. A Collaborative Virtual Environment for the Simulation of Temporal Bone Surgery. *Proceedings of MICCAI (Medical Image Computing and Computer-Aided Intervention) VII*, Rennes, France, September 26-30 2004. Springer-Verlag Lecture Notes in Computer Science Volumes 3216 and 3217.

[129] Morris D, Tan HZ, Barbagli F, Chang T, Salisbury K. Haptic Training Enhances Force Skill Learning. In review, August 2006.

[130] Morris D. Automatic Preparation, Calibration, and Simulation of Deformable Objects. Stanford University Department of Computer Science Technical Report 2006-07, August 2007.

[131] Morris D. TG2: A software package for behavioral neurophysiology and closed-loop spike train decoding. Technical documentation, 2006. Available at http://cs.stanford.edu/~dmorris/projects/tg2_description.pdf .

[132] Morris D. Algorithms and Data Structures for Haptic Rendering: Curve Constraints, Distance Maps, and Data Logging. Stanford University Department of Computer Science Technical Report 2006-06, June 2006.

[133] Mosegaard J and Sørensen TS. GPU Accelerated Surgical Simulators for Complex Morphology. *Proceedings of IEEE Virtual Reality 2005*.

[134] Mosegaard J, Herborg P, Sørensen TS. A GPU Accelerated Spring Mass System for Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality (MMVR) 13*, 2005.

[135] Mosegaard J. Parameter Optimisation for the Behaviour of Elastic Models over Time. *Proceedings of Medicine Meets Virtual Reality 12*, 2004.

[136] Mount DM and Arya S. ANN: A library for approximate nearest neighbor searching. *CGC 2nd Annual Fall Workshop on Computational Geometry*, 1997. Available at http://www.cs.umd.edu/~mount/ANN .

[137] Mueller M and Teschner M. Volumetric Meshes for Real-Time Medical Simulations. *Proc. BVM '03*, Erlangen, Germany, pp. 279-283, March 2003.

[138] Munich M and Perona P. Continuous Dynamic Time Warping for Translation-Invariant Curve Alignment with Applications to Signature Verification. *Proceedings of the 7th IEEE Conference on Computer Vision*, Kerkyra, Greece, 1999.

[139] Nakadi IE, Melot C, Closset J, DeMoor V, Betroune K, Feron P, Lingier P, Gelin M. Evaluation of da Vinci Nissen fundoplication clinical results and cost minimization. *World Journal of Surgery*, 2006, Jun;30(6):1050-4.

[140] Neumann P, Siebert D, Faulkner G, Krauss M, Schulz A, Lwowsky C, Tolxdorff T. Virtual 3D Cutting for Bone Segment Extraction in Maxillofacial Surgery Planning. *Proceedings of Medicine Meets Virtual Reality (MMVR) 1999*.

[141] Nicolau SA, Pennec X, Soler L, Ayache N. A complete augmented reality guidance system for liver punctures: first clinical evaluation. *Medical Image Computing and Computer-Assisted Intervention* (MICCAI), 2005;8(Pt 1):539-47.

[142] Nienhuys H-W and van der Stappen AF. A surgery simulation supporting cuts and finite element deformation. *Proceedings of Proceedings of Medical Image Computing and Computer Assisted Intervention (MICCAI) 2001*.

[143] NVIDIA Corporation. nvtristrip library. February 2004. http://developer.nvidia.com/ .

[144] Nvidia tutorial: http://developer.nvidia.com/object/skinning.html

[145] Ojakangas CL, Shaikhouni A, Friehs GM, Caplan AH, Serruya MD, Saleh M, Morris DS, Donoghue JP. Decoding movement intent from human premotor cortex neurons for neural prosthetic applications. To appear in the *Journal of Clinical Neurophysiology*, 2006.

[146] Ost D, DeRosiers A, Britt EJ, Fein AM, Lesser ML, and Mehta AC. Assessment of a Bronchoscopy Simulator. American Journal of Respiratory and Critical Care Medicine, Volume 164, Number 12, December 2001, 2248-2255.

[147] Pai DK, Lang J, Lloyd JE, and Woodham RJ. ACME, A Telerobotic Active Measurement Facility. *Proceedings of the Sixth International Symposium on Experimental Robotics*, Sydney, Australia, March 1999.

[148] Pai DK, van den Doel K, James DL, Lang J, Lloyd JE, Richmond JL, and Yau SH. Scanning Physical Interaction Behavior of 3D Objects. *Computer Graphics* (ACM SIGGRAPH 2001 Conference Proceedings), August 2001.

[149] Pal NR and Pal SK. A review on image segmentation techniques. *Pattern Recognition*, vol. 26, no. 9, pp. 1277-1294, 1993.

[150] Patton JL and Mussa-Ivaldi FA. Robot-Assisted Adaptive Training: Custom Force Fields for Teaching Movement Patterns. *IEEE Transactions on Biomedical Engineering*, Vol. 51, No. 4, 2004.

[151] Petersik A, Pflesser B, Tiede U, Höhne KH, Leuwer R, Rudolf Leuwer. Realistic haptic volume interaction for petrous bone surgery simulation.

*Proceedings of Computer Assisted Radiology and Surgery (CARS) 2002*. Springer-Verlag, Berlin, 2002, 252-257.

[152] Petersik A, Pflesser B, Tiede U, Höhne KH, Leuwer R. Realistic Haptic Interaction in Volume Sculpting for Surgery Simulation. *Proceedings of Surgery Simulation and Soft Tissue Modeling (IS4TM) 2003*. Springer-Verlag Lecture Notes in Computer Science 2673, Springer-Verlag, Berlin, 2003, 194-202.

[153] Petersik A, Pflesser B, Tiede U, Höhne KH, Leuwer R. Haptic volume interaction with anatomic models at sub-voxel resolution. *10th IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. Orlando, FL, 2002, 66-72.

[154] Petrakis EGM, Diplaros A, Milios EE. Matching and Retrieval of Distorted and Occluded Shapes Using Dynamic Programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 11, 2002.

[155] Pflesser B, Petersik A, Tiede U, Hohne KH, Leuwer R. Volume cutting for virtual petrous bone surgery. *Computer Aided Surgery* 2002;7(2):74-83.

[156] PhysX, Ageia, http://www.ageia.com/

[157] Pintilie G and McInerney T. Interactive Cutting of the Skull for Craniofacial Surgical Planning. *IASTED International Conference Biomedical Engineering 2003 (BioMed2003)*, Salzberg, Austria, June 2003.

[158] Pausch R, Burnette T, Brockway D, Weiblen M. Navigation and Locomotion in Virtual Worlds via Flight into Hand-Held Miniatures. *Proceedings of ACM SIGGRAPH 1995*.

[159] Raymaekers C, De Boeck J, Coninx K. An Empirical Approach for the Evaluation of Haptic Algorithms. *IEEE World Haptics 2005*, Pisa, Italy.

[160] Renz M, Preusche C, Potke M, Kriegel HP, Hirzinger G. Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. *Proceedings of Eurohaptics*, p149-154, 2001.

[161] Reznek MA, Rawn CL, and Krummel TM. Evaluation of the Educational Effectiveness of a Virtual Reality Intravenous Insertion Simulator. Academic *Emergency Medicine* (abstract) 2002, 9(11): 1319-1325.

[162] Richards C, Rosen J, Hannaford B, Pellegrini C, Sinanan M. Skills evaluation in minimally invasive surgery using force/torque signatures. *Surgical Endoscopy*. 2000, 14: 791-798.

[163] Rosen J, Hannaford B, Richards CG, Sinanan MN. Markov modeling of minimally invasive surgery based on tool/tissue interaction and force/torque signatures for evaluating surgical skills. *IEEE Transactions on Biomedical Engineering*. 2001, 48(5): 579-591.

[164] Rosen J, Solazzo M, Hannaford B, Sinanan M. Objective laparoscopic skills assessments of surgical residents using Hidden Markov Models based on haptic information and tool/tissue interactions. *Proceedings of Medicine Meets Virtual Reality (MMVR) 2001*.

[165] Ruffaldi E, Morris D, Edmunds T, Barbagli F, and Pai DK. Standardized Evaluation of Haptic Rendering Systems. *Proceedings of the 14th IEEE Haptics Symposium*, March 2006, Washington, DC.

[166] Sakata S. Asamin 1.30. 2006. http://www.econ.ubc.ca/ssakata/public_html/software/

[167] Sakoe H and Chiba S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(1):43—49, 2004.

[168] Samani A, Bishop J, Luginbuhl C, Plewes DB. Measuring the elastic modulus of ex vivo small tissue samples. *Physics in Medicine and Biology*, 48, July 2003.

[169] Scharstein D and Szeliski R. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1/2/3):7-42, April-June 2002.

[170] Schmidt JG, Berti G, Fingberg J, Cao J, Wollny G. A Finite Element Based Tool Chain for the Planning and Simulation of Maxillo-Facial Surgery. *Proceedings of the fourth ECCOMAS*, Jyvaskyla, Finland, 2004.

[171] Schneider P and Eberly DH. Geometric Tools for Computer Graphics. Morgan-Kauffman, 2003. Relevant source:
http://www.geometrictools.com/Foundation/Distance/Wm3DistVector3Segment3.cpp
http://www.geometrictools.com/Foundation/Distance/Wm3DistVector3Triangle3.cpp

[172] Sederberg TW and Parry SR. Free-Form Deformation of Solid Geometric Models. *Proceedings of SIGGRAPH '86*, Computer Graphics 20, 4 (August 1986), 151-159.

[173] SensAble Technologies, Inc. OpenHaptics toolkit. http://www.sensable.com

[174] SenseGraphics AB. H3D API. http://www.h3d.org/

[175] Sethian JA. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, volume 93 of 4, pages 1591-1595, 1996.

[176] Sewell C, Morris D, Blevins N, Barbagli F, and Salisbury K. A Simulator for Training a Full Mastoidectomy Procedure. Demo paper presented at *IEEE World Haptics*, Pisa, Italy, March 2005.

[177] Sewell C, Morris D, Blevins N, Barbagli F, and Salisbury K. Achieving Proper Exposure in Surgical Simulation. *Proceedings of MMVR (Medicine Meets Virtual Reality) XIV*, Long Beach, CA, January 2006. Studies in Health Technology and Informatics, Volume 119.

[178] Sewell C, Morris D, Blevins N, Barbagli F, and Salisbury K. An Event-Driven Framework for the Simulation of Complex Surgical Procedures. *Proceedings of MICCAI (Medical Image Computing and Computer-Aided Intervention) VII*, Rennes, France, September 26-30 2004. Springer-Verlag Lecture Notes in Computer Science Volumes 3216 and 3217.

[179] Sewell C, Morris D, Blevins N, Barbagli F, and Salisbury K. Quantifying Risky Behavior in Surgical Simulation. *Proceedings of MMVR (Medicine*

*Meets Virtual Reality) XIII*, Long Beach, CA, January 2005. Studies in Health Technology and Informatics, Volume 111.

[180] Seymour NE, Gallagher AG, Roman SA, O'Brien MK, Bansal VK, Andersen DK, Satava RM. Virtual reality training improves operating room performance. *Annals of Surgery* 2002, 236(4): 458-464.

[181] Si H. TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, v1.3 User's Manual. Weierstrass Institute for Applied Analysis and Stochastics, Technical Report No. 9, 2004.

[182] SimBionix, http://www.simbionix.com

[183] SimSurgery, http://www.simsurgery.com/

[184] Srimathveeravalli G and Thenkurussi K. Motor Skill Training Assistance Using Haptic Attributes. *Proceedings of the First Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (WorldHaptics), Pisa, Italy, 2005.

[185] Srivastava S, Youngblood P, Rawn C, Hariri S, Heinrichs L, Ladd A. Initial evaluation of a shoulder arthroscopy simulator: Establishing construct validity. *Journal of Shoulder & Elbow Surgery*, Mar-Apr; 13(2): 196-205.

[186] Stam J. Real-Time Fluid Dynamics for Games. *Proceedings of the Game Developer Conference*, March 2003.

[187] Stoakley R, Conway M, Pausch R. Virtual Reality on a WIM: Interactive Worlds in Miniature. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI) 1995*.

[188] Stoev S and Schmalstieg D. Application and Taxonomy of Through-the-Lens Techniques. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, 2002.

[189] Sud A, Otaduy M, Manocha D. DiFi: Fast 3D Distance Field Computation Using Graphics Hardware. *Eurogrpahics* 2004.

[190] Sudarsanam N, Grimm C, Singh K. Intuitive tools for camera manipulation. Technical Report 2004-84, Washington University in St. Louis, 2004.

[191] Surgical Science, http://www.surgical-science.com/

[192] Tan D, Robertson G, Czerwinski M. Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting. *Proceedings of the Conference on Human Factors in Computing Systems (CHI) 2001*.

[193] Tan HZ. Identification of sphere size using the PHANToM: Towards a set of building blocks for rendering haptic environments. *Proceedings of the ASME Annual Meeting*, Vol. 61, Nov 1997.

[194] Tay B, Stylopoulos N, De S, Rattner DW, Srinivasan MA. Measurement of In-vivo Force Response of Intra-abdominal Soft Tissues for Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality*, pages 514-519, Newport Beach, January 2002.

[195] Tejada E and Ertl T. Large Steps in GPU-based Deformable Bodies Simulation. *Simulation Theory and Practice*, Special Issue on Special Issue on Programmable Graphics Hardware. 2005.

[196] Teschner M, Girod S, Girod B. Interactive Osteotomy Simulation and Soft-Tissue Prediction. *Proceedings of Vision, Modeling, and Visualization (VMV ) 1999*. Erlangen, Germany, pp. 405-412, Nov. 1999.

[197] Teschner M, Girod S, Girod B. Interactive Simulation Environment for Craniofacial Surgery Planning. *Proceedings of Computer Assisted Radiology and Surgery (CARS) 2000*. San Francisco, USA, pp. 51-56, June 28 - July 1, 2000.

[198] Teschner M, Girod S, Girod B. Optimization Approaches for Soft-Tissue Prediction in Craniofacial Surgery Simulation. *Proceedings of Medical Image Computing and Computer-Assisted Intervention (MICCAI) 1999*. Cambridge, England, pp. 1183-1190, Sep. 19-23, 1999.

[199] Teschner M, Heidelberger B, Mueller M, Gross M. A Versatile and Robust Model for Geometrically Complex Deformable Solids. *Proceedings of Computer Graphics International* (CGI'04), Crete, Greece, pp. 312-319, June 16-19, 2004.

[200] Tholey G, Desai J, Castellanos A. Force Feedback Plays a Significant Role in Minimally Invasive Surgery. *Annals of Surgery*, Vol. 241, No. 1, 2005.

[201] Thrane N and Simonsen LO. A comparison of acceleration structures for GPU assisted ray tracing. Master's thesis, University of Aarhus, Denmark, 2005

[202] Trichili H, Bouhlel MS, Kammoun F. A review and evaluation of medical image segmentation using methods of optimal filtering. *Journal of Testing and Evaluation*, vol. 31, no. 5, pp. 398-404, 2003.

[203] Van Gelder A. Approximate Simulation of Elastic Membranes by Triangle Meshes. *Journal of Graphics Tools*,3:21--42, 1998, 178-180.

[204] Van Sickle KR, McClusky DA 3rd, Gallagher AG, Smith CD. Construct validation of the ProMIS simulator using a novel laparoscopic suturing task. *Surgical Endosccopy* 2005; 19: 1227-1231.

[205] Varadhan G, Krishnan S, Sriram T, Manocha D. Topology Preserving Surface Extraction Using Adaptive Subdivision. *Eurographics Symposium on Geometry Processing*, 2004.

[206] Wagner C, Stylopoulos N, Howe R. The Role of Force Feedback in Surgery: Analysis of Blunt Dissection. *Proceedings of the 10th IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, March 2002.

[207] Wang T, Darzi A, Foale R, and Schilling R. Virtual Reality Permanent Pacing: Validation of a Novel Computerized Permanent Pacemaker Implantation Simulator. *Journal of the American College of Cardiology* (Supplement) 2001; 37(2): 493A-494A.

[208] Ware C and Osborne S. Exploration and Virtual Camera Control in Virtual three Dimensional Environments. *Proceedings of the 1990 symposium on Interactive 3D Graphics (I3D)*.

[209] Webster R, Haluck RS, Zoppetti G, Benson A, Boyd J, Charles N, Reeser J, Sampson S. A Haptic Surgical Simulator for Laparoscopic Cholecystectomy using Real-time Deformable Organs. *Proceedings of the IASTED International Conference Biomedical Engineering*. June 25-27, 2003, Salzburg, Austria, 2003.

[210] Webster RW, Haluck R, Mohler B, Ravenscroft R, Crouthamel E, Frack T, Terlecki S, Sheaffer J. Elastically deformable 3d organs for haptic surgical simulators. In *Proceedings of Medicine Meets Virtual Reality* (MMVR) 2002.

[211] Williams RL, Srivastava M, Conaster R, Howell JN. Implementation and Evaluation of a Haptic Playback System. *Haptics-e*, Vol. 3, No. 3, May 3, 2004.

[212] Wong T, Darzi A, Foale R, Schilling RJ. Virtual reality permanent pacing: validation of a novel computerized permanent pacemaker implantation simulator. *Journal of the American College of Cardiology* 2001; (Suppl)37(2):493A-4A.

[213] Xitact, http://www.xitact.com/

[214] Yokokohji Y, Hollis R, Kanade T. Toward Machine Mediated Training of Motor Skills: Skill Transfer from Human to Human via Virtual Environment. 1996. *Proceedings of the 5th IEEE International Workshop on Robot and Human Communication*.

[215] Youngblood P, Srivastava S, Curet M, Heinrichs W, Dev P, Wren S. Comparison of training on two laparoscopic simulators and assessment of

skills transfer to surgical performance. *Journal of the American College of Surgeons,* Volume 200, Issue 4, Pages 546-551. April 2005.

[216] Zeleznik R and Forsberg A. UniCam – 2D Gestural Camera Controls for 3D Environments. *Proceedings of the 1999 Symposium on Interactive 3D Graphics (I3D)*.

[217] Zhang H. Mesh Generation for Voxel-Based Objects, PhD Thesis, West Virginia University, 2005.

[218] Zilles CB and Salisbury JK. A Constraint-based God-object Method for Haptic Display. *International Conference on Intelligent Robots and Systems*, 1995.