Volume-based Graphics and Haptics Rendering Algorithms for Immersive Surgical Simulation

BY

SILVIO RIZZI

B.S., Electronics Engineering, Universidad Tecnológica Nacional, Argentina, 2002
M.S., Electrical and Computer Engineering, University of Illinois at Chicago, 2006

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Industrial Engineering and Operations Research
in the Graduate College of the
University of Illinois at Chicago, 2013

Chicago, Illinois

Defense Committee:

P. Pat Banerjee, Chair and Advisor
Cristian Luciano,
David He,
Ali Alaraj, Neurosurgery
Ben Roitberg, University of Chicago

To my wife, Ana,
for her unconditional support.

ACKNOWLEDGEMENTS

I would like to thank my academic advisor, Prof. Pat Banerjee for his support, guidance, and the freedom granted to pursue the ideas in this thesis. I would also like to thank Prof. Cristian Luciano for these years of inestimable advice, sincere friendship, and countless hours of solid work invested in helping me bring these ideas to fruition.

My gratitude is also extended to the other members of my thesis committee, Prof. David He, Dr. Ali Alaraj and Dr. Ben Roitberg. They always managed to accommodate my questions in their busy schedules, providing, with no exceptions, feedback of the utmost quality.

I would also like to recognize Dr. Jaime Gasco for his tireless enthusiasm in developing the discipline of surgical simulation to its full potential. My work has greatly benefited from collaboration and insightful discussions with him.

Special thanks to my Director of Graduate Studies, Prof. Michael Scott, and to Iris, Veronica, Evelyn, Alan and Monica at the UIC Department of Mechanical and Industrial Engineering.

The funding provided by US Department of State-Fulbright, University of Illinois at Chicago, and ImmersiveTouch, Inc is acknowledged with appreciation.

SR

TABLE OF CONTENTS

CHAPTER **PAGE**

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF ABBREVIATIONS

3D	Three dimensional
AANS	American Association of Neurological Surgeons
AR	Augmented Reality
CNS	Congress of NeuroSurgeons
CT	Computed Tomography
DICOM	Digital Imaging and Communications in Medicine
DOF	Degrees of Freedom
FOM	Foramen of Monro
FPS	Frames per Second
GPU	Graphics Processing Unit
ITK	Insight Segmentation and Registration ToolKit
MRI	Magnetic Resonance Imaging
OR	Operating Room
STL	Stereo Litography
VHTK	Volume Haptics ToolKit
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
VTK	Visualization ToolKit
X3D	Extensible 3D Graphics

SUMMARY

This work shows the research and development involved in solving essential problems in the emerging field of surgical simulation. It focuses on a haptics-based Augmented Reality surgical simulation platform known as *ImmersiveTouch*®, which implements technologies patented by the Board of Trustees of the University of Illinois.

Through the nine chapters of this thesis, a gradual transition to new paradigms in surgical simulation is naturally developed, starting from methods to create patient-specific 3D models for training and pre-operative planning, continuing with the development of a voxel-based haptics algorithm, its performance evaluation, and extensions for multipoint collision detection; followed by the introduction of graphics and haptics techniques that are combined to simulate bone-removal procedures, and culminating in the successful implementation of surgical simulation modules on the *ImmersiveTouch*®

Multiple validation experiments are also presented, where some of the contributions in this thesis are used in simulation modules that are evaluated in surgical training scenarios with promising and encouraging outcomes. Multi-disciplinary collaboration is one of the highlights of this work, with scientifically sound results published in prestigious peer-reviewed engineering and medical journals and conferences.

1. INTRODUCTION

ImmersiveTouch®, the surgical simulator platform used in this thesis (Luciano et al., 2005; Banerjee et al., 2010), consists of multiple hardware and software components, including collocated 3D graphics and haptics, head and hand tracking, and a Software Development Kit (SDK) that integrates a number of software libraries, including:

- Coin3D, as an open implementation of OpenInventor for scene graph management
- FLTK, for graphical user interfaces
- OpenHaptics, to interact with Sensable haptic devices
- OpenAL, to provide 3D audio
- PhysX, for dynamics-based simulation

In terms of 3D graphics, there are two well-established paradigms for data visualization: polygonal mesh rendering and volume rendering. Of these two, the simulator originally supported only polygonal mesh models for simultaneous haptics and graphics rendering. A crucial observation in the early stages of this research was that the simulator could be greatly enhanced by also supporting volumetric datasets. This was mainly motivated by the fact that CT, MRI, and other patient-specific datasets are essentially delivered as a discrete grid representation in space of physical magnitudes (magnetic field intensity, x-ray intensity, etc.). Therefore, by directly supporting these voxel-based datasets, the necessity of converting patient data to polygonal meshes could be minimized, or even entirely avoided.

In this way, graphics volume rendering was incorporated into the simulator SDK by adding the SimVoleon volume rendering library to Coin3D. A voxel-based haptics library was available (Lundin et al., 2006), but its combined performance with SimVoleon was

sub-optimal within the ImmersiveTouch framework. This fact pointed our research to developing an alternative volume haptics algorithm. It was additionally found that the combination of this novel volume haptics algorithm with polygonal mesh visualization yielded the best performance on the ImmersiveTouch. The polygonal mesh models necessary for graphics rendering could be obtained from voxel models by using a well-known algorithm for polygonization of scalar fields (Marching Cubes). From that point, additional breakthroughs followed, such as the implementation of material-removing algorithms for simulation of bone surgery or the development of extensions to the haptics algorithm for object-to-object collision detection.

This Ph.D. dissertation consists of nine chapters. Chapter 2 presents a method for segmenting anatomies of interest from medical images preserving their spatial continuity and coherence, resulting in high-quality polygonal meshes with a low number of polygons that are optimal for simultaneous haptics and graphics simulation. Chapter 3 describes a haptic algorithm able to generate force-feedback from voxels, without the need of generating polygonal mesh representations of the 3D models. In Chapter 4, results of multiple experiments evaluating the performance of existing haptic algorithms are presented, where it is demonstrated that the combination of polygonal mesh graphics rendering with volume haptics rendering provides the best performance for surgical simulation applications. Based on the haptics algorithm from Chapter 3, extensions for multipoint collision detection are introduced in Chapter 5. With that, a fundamental limitation in existing haptic libraries (i.e interaction with only a single point) is overcome. Chapter 6 introduces yet more fundamental advances, in which the previously discussed optimal combination of polygonal mesh graphics rendering with volume

haptics rendering is augmented with the capability of rapidly regenerating the graphics polygonal mesh. Algorithms for simulation of burr-hole drilling, skin incisions, and cutting of a craniotome are also presented in Chapter 6. Practical implementations of the algorithms are shown in Chapter 7, where simulation modules for ventriculostomy with burr-hole drilling, percutaneous spine needle insertion, and subclavian central line are described. Chapter 8 presents the results of four experiments validating different simulation modules in which the contributions of this thesis have been used. Finally, Chapter 9 summarizes the major contributions of this work.

2. CREATING MODELS FOR THE IMMERSIVETOUCH

2.1 The necessity of segmentation, model generation and additional processing

The ImmersiveTouch simulator (Luciano et al., 2005; Banerjee et al., 2010) is the latest generation of augmented Virtual Reality (VR) technology, which integrates a haptic device with a head and hand tracking system, and a high-resolution and high-pixel-density stereoscopic display. A haptic device collocated with 3D graphics is the key factor to deliver extremely realistic simulations. Previously, the ImmersiveTouch simulator has been successfully applied to the simulation of neurosurgical procedures and training of neurosurgery residents (Luciano et al., 2006). It implements graphics and haptics rendering in a multi-threaded environment. In order to satisfy the minimum required graphic and haptic frame rates, it is essential to use efficient 3D models of the anatomical parts to be simulated.

In the context of on-demand high fidelity simulations (Banerjee, Charbel, 2006), automatic or semi-automatic techniques to generate 3D models from medical images are highly desirable. Segmentation techniques are applied to Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) images to obtain 3D models from medical data. These models have to be reduced and converted to polygonal surfaces for simultaneous graphics and haptics rendering. Depending on the application, further processing may be needed (e.g. drilling of burr holes for simulation of a neurosurgery procedure known as ventriculostomy).

2.2 From MRI/CT scans to polygonal models

The ImmersiveTouch is capable of simultaneous haptics and graphics rendering of 3D models. As an important feature of its software design, several models representing different organs or objects can be loaded at the same time. This requires implementing a sophisticated collision detection mechanism and the assignment of different haptic properties to each model. For medical applications, the models are extracted from MRI and/or CT scans. Collections of segmented images are converted to 3D volumes, to be further transformed into polygonal surfaces. It is important not only to obtain accurate 3D models, but also to make them efficient for graphics and haptics rendering. A frame rate of 60 Hz is used for stereoscopic graphics rendering, whereas a rate of 1000 Hz is used for haptics rendering. Polygonal meshes obtained from segmentation must be carefully decimated; otherwise the number of polygons would be excessively high, making it impossible for the haptics library to perform adequately at the above frame rate. A block diagram of the system is shown in Figure 1.

2.3 Research Problem:

Find an optimal way to implement the Pre-processing block in Figure 1 to extract anatomies of interest from multiple 2D slices and convert them into 3D polygonal meshes with a sufficiently low polygon count to allow interactive graphics and haptics rendering

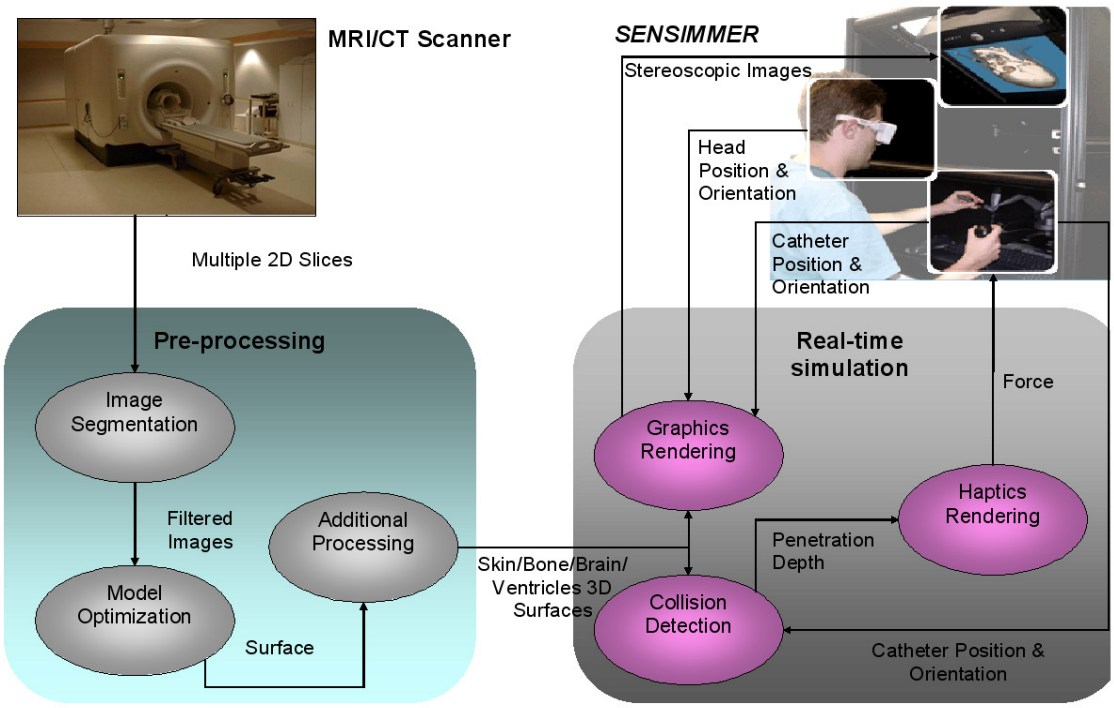


Figure 1. Block diagram of the system.

2.4 Previous work

Several systems and methods to address similar problems have been reported in the literature. Cebral and Löhner (2001) proposed a method for construction of arterial surface models from medical images. A region-growing algorithm was used to segment the arteries. After isosurfacing, smoothing, and mesh optimization, a finite element mesh suitable for computational fluid dynamics calculations was generated. User interaction was required during the segmentation and geometry modeling stages. Even though some algorithms and methods were discussed, and few examples where the method succeeds were shown, specific software implementations were not presented. The time required to perform the entire process was reported to be in the order of few hours for the examples shown

Du et al. (2005) presented an integrated system based on ITK and VTK. Their focus was on creating finite elements models from CT or MRI images, but specific applications were not shown. Segmentation was done either manually or automatically. However, it is not clear how the system determined the regions to be segmented when working in automatic mode. Besides that, it was neither specified how much user interaction is required nor the time needed to perform the entire process for common cases.

Ito et al. (2006) proposed a method for unstructured mesh generation for high-fidelity numerical simulations. Their main goal was to obtain high quality surface meshes. For that, two different approaches were used: direct advancing front method and modified decimation method. ITK and VTK were used along with custom code. Segmentation was done using threshold filters and transfer functions. The authors remarked the importance of working closely with medical experts to validate the results of segmentation.

Young et al. (2006) presented examples where 3D image data is automatically converted into polygonal meshes. Since the approach used was outlined but not detailed, it is difficult to evaluate the degree of automation achieved. Also, the degree of human intervention required was not discussed.

Melonakos et al. (2005) presented an implementation in ITK of a segmentation algorithm based on incorporating prior knowledge through Bayes' rule. The intensity value of each voxel was considered a random variable. Additional assumptions on intensity distributions and prior likelihoods were made. According to the authors, this knowledge-based segmentation algorithm required minimal user interaction. Examples of two different applications where the algorithm succeeds were shown.

Wolf et al. (2004) described The Medical Imaging Interaction Toolkit (MITK), an object-oriented, cross-platform library extending VTK and ITK. According to the authors, their goal was not to reinvent anything already existing, but to add new features to the previous development. It was stressed the fact that software for clinical use in image-guided procedures and image analysis required a high degree of interaction to verify and correct results from automatic or semi-automatic algorithms.

2.5 Software tools

We have identified two software packages that are promising for the tasks of image segmentation and model optimization required in Figure 1: ITK and VTK

ITK (The Insight Segmentation and Registration Toolkit) is an application framework initially developed to support a U. S. National Library of Medicine's project (The Visible Human Project). In addition, (ITK-SNAP) is an open-source software package, built on top of ITK, oriented to the segmentation of 3D anatomical structures from medical images. Using ITK-SNAP, it is possible to perform segmentation as a semi-automated procedure. Though referred as snakes (Kass et al., 1997) within the software, ITK-SNAP uses two 3D active contour segmentation methods (Yushkevich et al., 2005), namely Geodesic Active Contours (Caselles et al., 1997) - driven by intensity edges - and Region Competition (Zhu, Yuille, 1996) - driven by intensity regions. ITK-SNAP has been validated as a highly reliable tool in the context of a child autism neuroimaging study (Yushkevich et al., 2006).

VTK (The Visualization ToolKit) is an open-source software package for visualization that supports a wide variety of advanced visualization and volume

processing algorithms. In VTK, it is possible to construct visualization pipelines consisting of data and process objects (Schroeder et al., 2006).

In this context, we use ITK-SNAP for image segmentation and VTK for model optimization.

2.6 Methodology

2.6.1 Segmentation with ITK-SNAP

ITK-SNAP provides a friendly user interface by which the user guides the segmentation process in a semi-automatic manner. Its source code is freely available, it is being actively developed, and it has a growing community of users.

Series of CT images in DICOM format are read by ITK-SNAP. Orthogonal axial, coronal, and sagittal planes are displayed. The user can adjust the image histogram to enhance the visualization and contrast of the anatomical part under study. As the first step in segmentation, the volumetric region on which to perform the segmentation is selected. Input images have to be preprocessed before being fed into the segmentation algorithm. ITK-SNAP provides two methods for image preprocessing: Intensity Regions and Image Edges. In our case, Intensity Regions is the method that gives best results. Essentially, the method consists of applying a thresholding function to the input images. Two probability fields are estimated: the probability that a pixel in the image belongs to the foreground (structure of interest), and the probability that the pixel belongs to the background. The active contour is attracted to the points where both probabilities are equal (Yushkevich et al., 2005).

Initially, the user must place 3D spheres of variable radius - called "bubbles" - as starting values for the algorithm. Afterwards, the active contour evolves continuously in every iteration, eliminating great part of the original noise. The output of this process is a subset of voxels from the original CT scan that are identified as part of the anatomies of interest.

2.6.2 Generation and optimization of 3D models with VTK

VTK is used to construct an optimal 3D representation of the data. Figure 2 describes the VTK pipeline used. The first stage consists of a vtkPDataSetReader filter, which reads the data segmented in ITK-SNAP and outputs volumetric data.

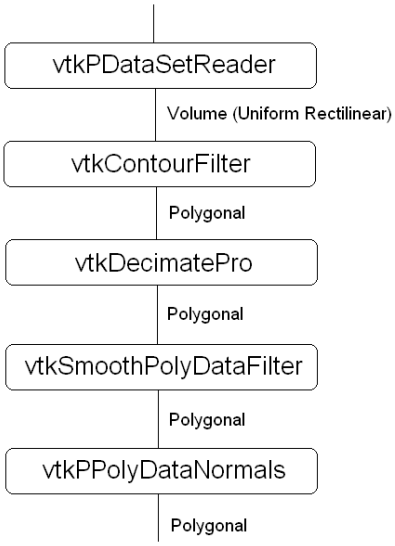


Figure 2. VTK pipeline.

The next filter is vtkContourFilter, which generates an isosurface from its input data. This isosurface must be drastically decimated to reduce the number of polygons to

be rendered. For that, a vtkDecimatePro filter is used with the option "preserve topology" activated.

Following decimation, a smoothing filter is applied using vtkSmoothPolyDataFilter. The number of iterations required for each model was determined observing the smoothness of the model. After that, a vtkPPolyDataNormals filter is applied to generate normal vectors for each polygon. This step is essential for a correct rendering of the resulting models. Finally, the polygonal mesh is saved in VTK file format to be further converted into VRML or a different 3D representation (e.g. Stereo Lithography - STL) should additional processing is needed.

2.7 Results

To test the tools previously described, we created a model of a challenging case of ventriculostomy corresponding to a patient whose ventricles are notoriously shifted. For this experiment, the input data consists of 192 CT images in DICOM format. Each image contains intensity levels of 512 by 512 pixels. Intensity levels are given as 16-bit signed integers.

For the existing ventriculostomy simulator (Luciano et al., 2006), there are four models to be created from patient data: ventricles, brain, skull, and skin. ITK-SNAP was used to segment these anatomies. An example of the shifted ventricles after segmentation is shown in Figure 3:

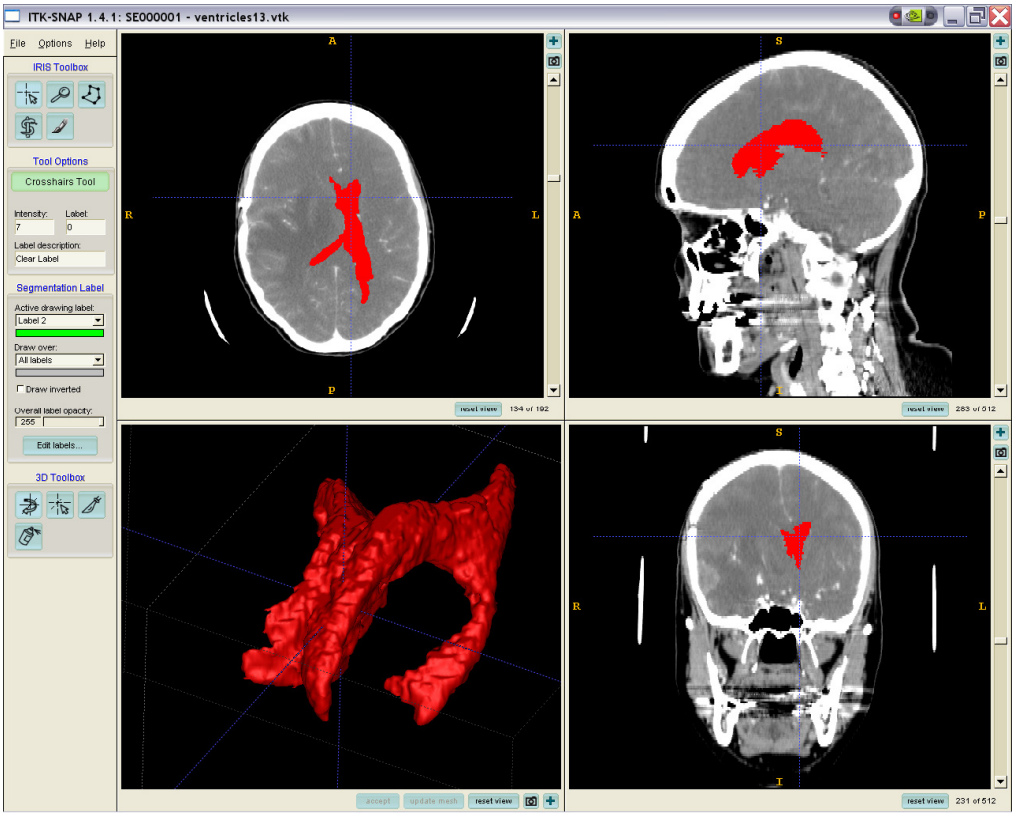


Figure 3. Ventricle segmentation in ITK-SNAP.

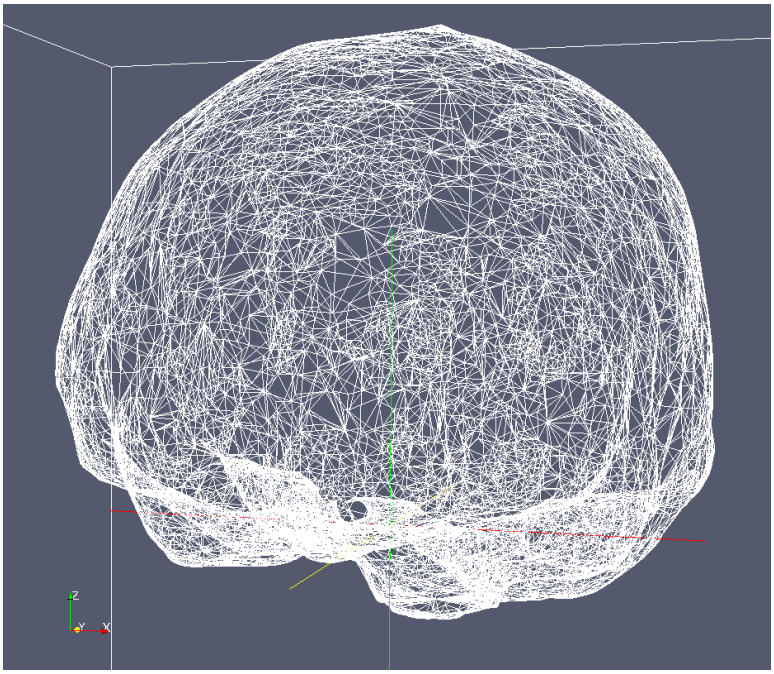


Figure 4. Polygonal mesh of the brain.

After the segmentation process, meshes are generated from the segmented voxels. Figure 4 shows an example of the brain polygonal mesh obtained as part of the 3D model generation stage:

A significant reduction in the number of polygons was achieved as part of the optimization stage. The following table shows the values obtained:

TABLE I

3D MODELS AND THEIR PROPERTIES

Model		Before decimation	After decimation	% reduction
Brain	Number of polygons	495364	22742	95.4
	Number of vertices	247748	11437	95.4
	Memory (MBytes)	13.872	0.822	94.0
Ventricles	Number of polygons	93528	23176	75.2
	Number of vertices	46782	11606	75.2
	Memory (MBytes)	2.621	0.837	68.0
Skin	Number of polygons	916608	83346	90.9
	Number of vertices	458306	41675	90.9
	Memory (MBytes)	25.666	3.003	88.3
Skull	Number of polygons	1495636	79810	94.7
	Number of vertices	747604	39691	94.7
	Memory (MBytes)	41.875	2.87	93.1

In ventriculostomy, burr holes must be drilled according to anthropometric measures (Prabhu et al., 2004). We simulate the process creating cylinders and ellipsoid-like shapes that are combined with the original model using Boolean operations in 3DS Max. Using this tool, not only can the user easily find an optimum view of the model, but also the proper placement and orientation of the cylinders and ellipsoid-like shapes are facilitated (Figure 5)

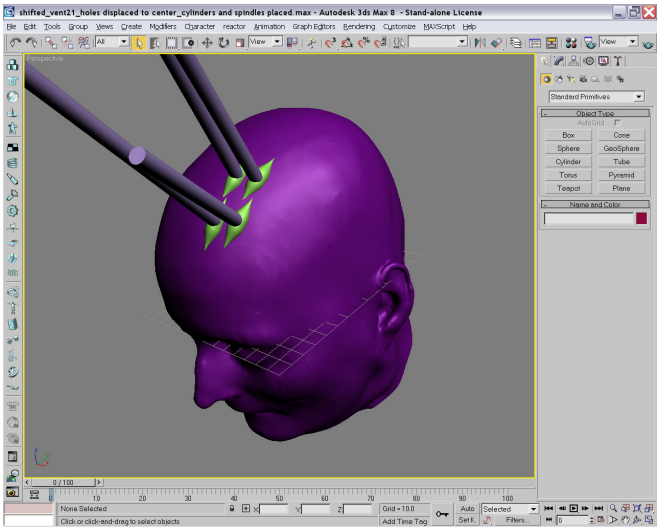


Figure 5. Finalizing models in 3DS Max.

A boolean(-) operation is performed between the skin model and the ellipsoid-like shapes. Similarly, a boolean(-) operation is also performed between the skull model and the cylinders. Before exporting the final models in VRML, 3DS Max is also used to edit their color, lighting and texture mapping. Different views of the final models are shown in the following figure:

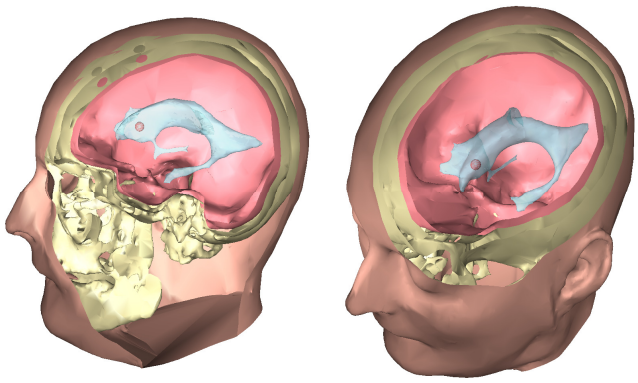


Figure 6. Final models.

2.8 Contribution

The procedures outlined in this section allow to define repeatable sequences of operations toward the automation of the 3D model creation process. Segmentation using the snakes algorithm in ITK-Snap results in 3D models preserving the continuity of the anatomies of interest, which is an essential feature for smooth simulation in the virtual environment. Significant reduction in the number of polygons (up to 95%) without affecting mesh quality is achieved with VTK.

The method has been successfully applied to the creation of a library of 15 cases for ventriculostomy simulation, including models of patients with normal, hydrocephalic, shifted and small ventricles. The library has been used as part of an experiment conducted at the Dr. Allan L. and Mary L. Graham Clinical Performance Center (CPC) at the University of Illinois-Chicago. Results have been reported in (Yudkowsky et al., 2010) and (Yudkowsky et al., 2012) and are summarized in Chapter 9.

3. VOLUME GRAPHICS AND HAPTICS

Lately, volumetric data sets have acquired extraordinary significance in medical simulation. Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) are examples of ubiquitous technologies from which 3-dimensional (3D) data sets are obtained. 3D computer models are commonly generated for Virtual Reality and Haptics simulation of medical and surgical procedures (Luciano et al., 2006; Banerjee, Charbel, 2006).

Traditionally, a combination of scene graph managers (Systems In Motion Coin3D; H3D) and haptic libraries (SensAble Technologies OpenHaptics) is used in simulations for simultaneous graphics and haptics rendering of 3D models. Those libraries commonly require 3D objects to be represented as polygonal meshes, i.e. surfaces in 3D space consisting of multiple triangles. These polygonal meshes are usually generated using an isosurface extraction algorithm such as Marching Cubes (Lorensen, Cline, 1987). Further processing may be required in order to reduce the number of triangles in each model (decimation) and to obtain smooth surfaces. All these processing stages often demand several hours -or even days- to complete, requiring the use of additional software tools and a considerable amount of human intervention to generate high quality 3D models. Although methods to accelerate and improve the degree of automation of the segmentation process have been discussed in the literature (Rizzi et al., 2007), alternative approaches must be explored in order to improve the simulations.

OpenHaptics (SensAble Technologies OpenHaptics) is one of the most popular commercial haptic libraries supporting SensAble haptic devices (Massie, Salisbury, 1994; SensAble Haptic Devices). It is extensively used in a number of systems for Haptics and

Virtual and Augmented Reality applications, including (H3D) and OpenHaptics-enabled versions of the software described in (Luciano et al., 2005; Banerjee et al., 2010). It has, however, serious limitations when it is required to haptically render highly complex shapes. Its two modes (Feedback Buffer and Depth Buffer) impose their own constraints on the model to be rendered. On one hand, Feedback Buffer delivers high quality haptic rendering, however its performance is dependent on the number of polygons in the model. On the other hand, Depth Buffer is insensitive to the number of polygons, but there are some cases where it exhibits "noticeable discontinuities when feeling shapes with deep, narrow grooves or tunnels" (SensAble OpenHaptics Toolkit Version 3.0 Programmer's Guide [a]).

To overcome the limitations discussed above, a more natural and straightforward approach would be to implement a direct volume haptics algorithm. Volumetric data could then be used directly as delivered by imaging systems, reducing or even eliminating the need of preprocessing stages to build models as polygonal meshes. Furthermore, a robust direct volume haptics could be a viable alternative to address those problems where OpenHaptics fails.

3.1 Research Problem:

Design and implement a Volume Haptics Algorithm that uses available haptics libraries to overcome inefficiencies of currently existing solutions

3.2 Overview

A volume haptics algorithm based on proxy methods is presented. It essentially consists of detecting collisions between the proxy point and one or more 3D shapes representing objects of interest. Shapes are defined from a set of voxels using transfer functions without the need to generate polygonal meshes.

The algorithm receives two points as parameters (Start and End) for each haptic frame rendered by the servoloop at 1 KHz. The Start point is the proxy position calculated in the previous haptic frame whereas the End point is the current position of the haptic device.

The collision detection routine detects the intersection between a line segment (determined by Start and End) and a shape surface (Figure 7). Shape surfaces are defined in terms of voxel intensities, similar to isosurfaces. The algorithm returns the 3D coordinates of the intersection point P, the normal vector N of the surface at the intersection point, and the touched side (front or back) of the shape surface. It also returns TRUE if there is a collision or FALSE otherwise. With this information, the underlying haptic library computes the forces as in the case of polygonal mesh haptics, and positions the proxy at the point P when a collision with the shape is detected.

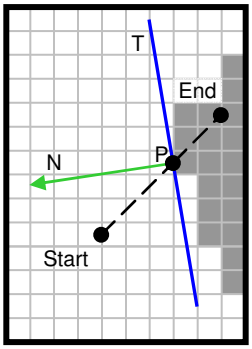


Figure 7. Problem geometry.

3.3 Algorithm Details

Figure 8 shows a flow diagram of the algorithm. If the haptic stylus has not moved in two successive haptic frames and there was no collision in the previous frame, then the Start and End points are exactly the same and therefore, the function returns FALSE. Otherwise, it continues with a rough bounding-box comparison between the line and the volume, quickly returning false if they are disjoint.

If the line is inside the volume bounding box, for each point P on the line segment from the Start to the End points, the algorithm checks the intensity to the closest voxel V by a set of window transfer functions defining the multiple shapes. If the voxel intensity is outside the windows specified through the transfer functions, then the haptic device has not yet collided with any shape and the loop continues with the next point. If none of the points on the line segment collide with any shape, the function returns false.

In case the intersected voxel V lies within any of the transfer function windows, the algorithm returns the 3D coordinates of the point P as the surface contact point. The density of the voxel V is used to determine which of the shapes has been touched by the haptic device by comparing it with the ranges defined by their transfer functions.

The normal vector N, which is perpendicular to the volumetric isosurface at the contact point P, is determined by computing the gradient of the neighbor voxels using the central differences method. The contact point P and the normal vector N define a plane T (tangential to the shape) which serves as an intermediate representation of the isosurface. This plane is useful to determine if the haptic device is touching either the front or back side of the shape. If the Start point is in front of the plane T and the End point is behind

it, then the colliding face is front. Otherwise, the colliding face is back. In this case, the algorithm inverts the direction of the previously computed normal vector N.

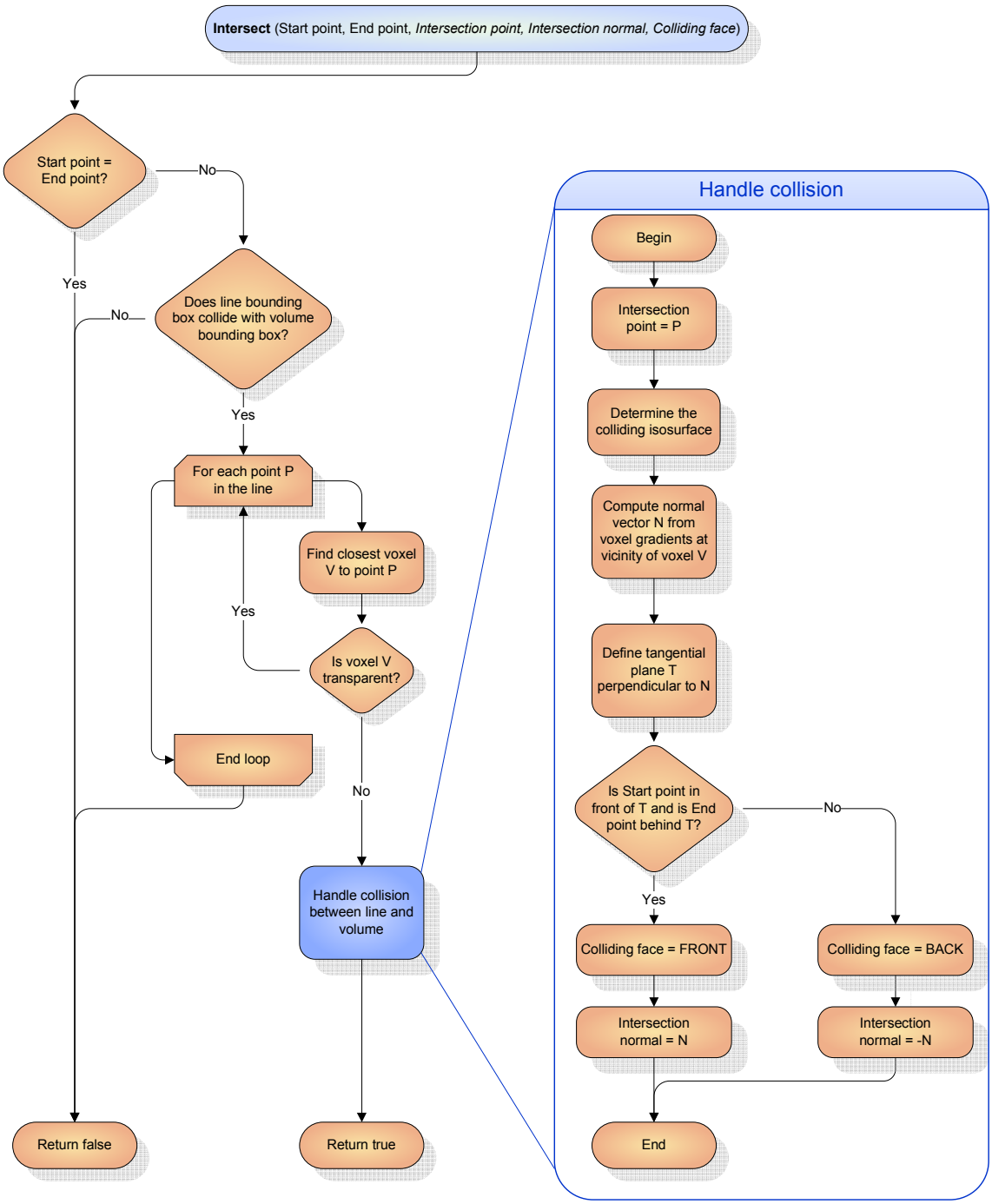


Figure 8. Flow diagram of the algorithm.

3.4 Algorithm Parameters

There are two essential design requirements for this algorithm. First of all, it must be efficient enough to not affect the performance of the haptics rendering servo loop sustaining a minimum 1 KHz haptic frame rate. In addition, it must be robust enough to avoid undetected collisions. Both requirements are directly affected by the selection of the step size with which successive discreet points along the line segment (shown in Figure 7) are evaluated. If the step size is too big, a collision could be overlooked, especially for thin structures. On the other hand, a finely grained step size could help guarantee collisions are always detected, but it could also severely impact the haptic frame rate, since the algorithm is executed in the servo loop thread.

If we parametrize the line segment from Start to End with parameter i, where i is in the interval [0,1], then the following linear interpolation equation gives any point P in the line segment as a function of i:

$$\mathbf{P} = (1 - i) \cdot \mathbf{Start} + i \cdot \mathbf{End} \qquad (1)$$

The algorithm traverses the line segment by varying i from 0 to 1, incrementing it by a value of delta in each successive iteration. Computations of P are done in continuous space and further converted to discrete voxel coordinates for retrieving voxel values. No sub-voxel resolution is needed.

Users can move the haptic stylus at various speeds, which is reflected in corresponding variations of the line segment length from Start to End. Therefore, delta must be carefully selected each time the algorithm is executed. A naïve approach would

be to divide the line segment into a constant number of steps, so the number of iterations is constant for all moving speeds. However, this approach would fail to detect collisions when the haptic device is moved at high speeds, especially when structures are thin. The problem is solved using a variable step size as follows:

$$delta = \frac{k}{|\mathbf{End} - \mathbf{Start}|} \qquad (2)$$

where k is a constant that depends on the voxel size. In this way, for higher speeds, the interval [0,1] representing the line segment is divided into a higher number of steps. As shown in Figure 8, when the haptic device does not move in two successive haptic frames (Start = End), the algorithm returns immediately, preventing division by zero in Equation (2). Every time the algorithm is executed, the actual distance between successive points P to be evaluated in a given line segment is constant, regardless of the velocity of the haptic stylus. Thus, initializing k to be equal to or less than the minimum dimension of voxels is a necessary condition to prevent undetected collisions.

3.5 Transfer Functions

The algorithm is able to simultaneously detect multiple shapes from volumetric isosurfaces defined by their individual ranges of voxel intensities. A transfer function is defined for each haptic shape whereby a binary output value is assigned to every possible voxel intensity. In graphics volume rendering techniques, piece-wise linear transfer functions are commonly used to specify color intensities and transparency. Similarly, in

our approach transfer functions are used to determine whether a voxel should be touchable or not based on its intensity.

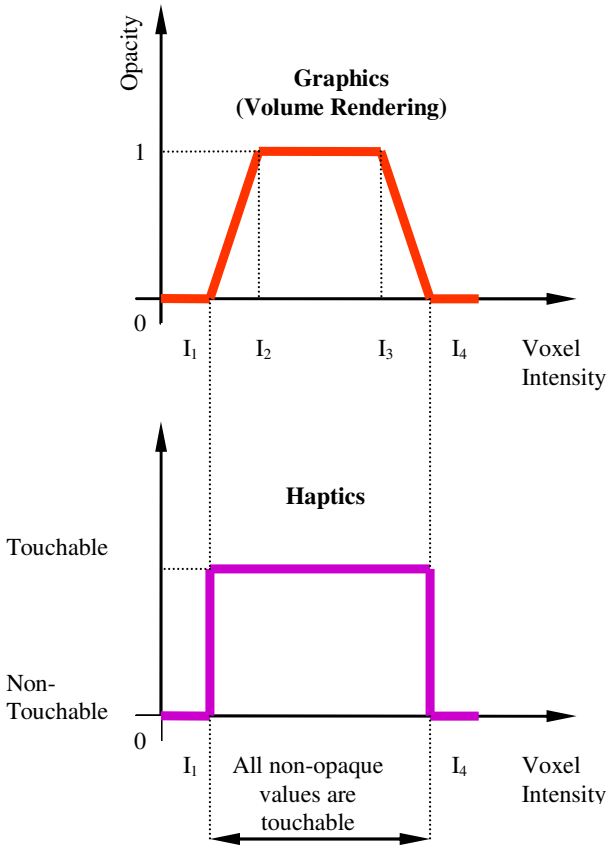


Figure 9. Graphics and Haptics transfer functions.

Figure 9 presents a comparison between graphics volume rendering and haptic transfer functions. The first transfer function exemplifies opacity as a function of voxel intensities. Gradually increasing or decreasing values of opacity, represented by ramps, are allowed and commonly used. On the other hand, in the haptics transfer functions only discrete binary outputs are permitted. In this way, voxels with intensities within the rectangular window defined by the transfer function will be regarded as belonging to the shape and will, therefore, be touchable. In other words, when the collision detection

algorithm finds a voxel whose intensity is within the rectangular window, it will return TRUE, indicating a collision with the shape was detected.

There are two advantages to using haptic transfer functions. First, since they are similar to the ones commonly used in volume visualization techniques, a single transfer function may simultaneously specify graphics and haptics properties for each shape. In Figure 10 it is shown how a haptic transfer function can be obtained from its graphics counterpart. As a result, all non-opaque values will be touchable and haptic parameters such as stiffness, static friction, and dynamic friction will be assigned to the corresponding voxels. The second advantage is that pre-processing steps such as segmentation and construction of polygonal meshes for each shape are no longer needed. In essence, the haptic transfer functions implemented resemble an operation of binary thresholding, by which different subsets may be determined from the original dataset according to their voxel intensities. Therefore, the specification of transfer functions provides all the information needed to generate graphics and haptics visualization, operating only with the original (unmodified) 3D dataset.

3.6 Implementation

The algorithm is intended to take advantage of the efficient force computation implemented in existing haptic libraries (i.e. Open Haptics). Therefore, the algorithm performs the collision detection and passes to the haptic library all the information needed to compute the forces in the same way it does for polygonal meshes.

OpenHaptics allows users to define custom shapes by a callback function which is called in each frame of the servoloop thread, before computing the forces to be sent to the

haptic device. The prototype of the intersect callback function for OpenHaptics (SensAble OpenHaptics Toolkit Version 2.0 Programmer's Guide [b]) is as follows:

```
bool intersectSurface(const HLdouble startPt[3],
                      const HLdouble endPt[3],
                    HLdouble intersectionPt[3],
                    HLdouble intersectionNormal[3],
                      HLenum *face,
                        void *userdata );
```

The algorithm, implemented as a callback function responding to the `intersectSurface` prototype, returns the coordinates of the contact point P, the intersection normal vector N, and the touching face (as the third, fourth and fifth parameters, respectively). OpenHaptics computes forces based on the haptic materials associated with the haptic shape (spring, damper, static and dynamic friction), allowing the user to feel the contact and friction between the proxy and the volumetric isosurfaces. Similar to the case of polygonal meshes, by setting the haptic shape's touchable face as HL_FRONT, HL_BACK or HL_FRONT_AND_BACK, the algorithm allows the user to feel only one or both sides of the haptic isosurface. If there is no collision, OpenHaptics updates the proxy position with the current position of the haptic device. On the other hand, if there is a collision, OpenHaptics fixes the proxy at the surface contact point P, and computes the forces to be sent to the haptic device.

To detect collisions with multiple shapes, OpenHaptics calls the `intersectSurface` callback function once for each haptic shape defined. Those multiple calls are made within each individual servo frame at 1 KHz. If the

`intersectSurface` callback function returns TRUE, there is a collision with the current shape. It returns FALSE, otherwise. However, the algorithm needs to be executed only once per haptic frame since the goal is to find the first non-transparent voxel along the line segment from Start to End. The sixth argument (`*userdata`) in the `intersectSurface` prototype is used to pass the shape to be evaluated in each call. The collision detection algorithm is executed only during the call corresponding to the first shape in a servo frame. If a collision is detected, the algorithm will determine the touched shape comparing the density value of the voxel against the value ranges of the transfer function. After the first call within a servo frame, there is no need to execute the collision detection algorithm again. However, based on the `*userdata` parameter, the callback function will return TRUE when the call corresponds to a collided shape, and FALSE for all the other shapes. In this way, OpenHaptics' `intersectSurface` callback function is called multiple times (once per existing shape) but the collision detection algorithm is executed only once in each frame of the servo loop, obtaining a constant runtime independent of the number of shapes.

3.7 Contributions

Comparing this algorithm with previous intermediate representation approaches, the major contributions follow.

3.7.1 Elimination of inadequate haptic feedback

For previous algorithms, such as (Adachi et al., 1995), there is a limitation where a lower update rate for the intermediate representation with respect to the servoloop rate

may cause irregularities in the force feedback. The problem is contemplated in the recovery time approach (Mark et al., 1996), but not eliminated. In (Chen et al., 2000), the update rate of the intermediate representation is 1/n of the force computation rate, and so the problem in (Adachi et al., 1995) is also present whenever n > 1. In our approach, collisions are detected at exactly the same rate in which the servoloop is updated, thus each execution of the collision detection is guaranteed to precede the force computation. Therefore, our algorithm eliminates this problem inherent in intermediate representations.

3.7.2 No fall-through for thin structures

The algorithm in (Chen et al., 2000) may fail to detect collisions with thin structures. This problem is not present in our algorithm, where the speed at which the haptic device is moved does not affect the robustness of the collision detection algorithm.

3.7.3 Haptic front/back face detection

Building our algorithm on top of an existing haptic library allows detection of back/front faces and to assign different haptic properties to each one. This is not possible in (Adachi et al., 1995) and (Chen et al., 2000).

3.7.4 Multiple shape detection

Our algorithm is implemented such that it is possible to efficiently detect multiple shapes and assign different haptic properties to each of them. This feature is not discussed in (Adachi et al., 1995) and (Chen et al., 2000).

3.7.5 Leveraging of existing libraries

Building our algorithm as part of an existing haptic library allows one to use volumetric as well as polygonal mesh models at the same time. Moreover, there are additional advantages from using the OpenHaptics library that come for free, such as pop-through effects as well as touch/untouch callback functions.

4. COMPARISON OF HAPTICS RENDERING ALGORITHMS

Existing haptics libraries present serious limitations when the complexity of the models is high. Specifically, OpenHaptics' Feedback Buffer mode can not deliver an adequate performance when 3D models are composed of a very high number of polygons. In addition, its Depth Buffer mode becomes unstable in regions of high curvature. Other volume haptics implementations, such as VHTK, suffer from fall-through and incorrect force computation. In this chapter, we will prove that these limitations are overcome by the algorithm presented in the previous chapter, which can be easily implemented as an extension to existing haptics libraries

4.1 Research Problem:

Compare haptics rendering algorithms combined with polygonal graphics rendering to assess the quality of haptic feedback provided, as well as identifying the best combination in terms of rendering time.

4.2 Overview

In modern graphics cards, graphic pipelines are optimized for polygonal mesh models. There is also a great majority of haptic algorithms based on polygonal meshes. On the other hand, a voxel-based approach for both graphics and haptics allows one to implement volume removal procedures with relative simplicity. However, graphics volume rendering techniques are slower than polygonal mesh graphics. For this reason, some researchers have opted for using voxel-based models for haptics and volume

removal procedures combined with polygonal meshes of the deformed models, which are generated on-the-fly (Morris et al., 2006). Moreover, results in (Rizzi et al., 2010) also suggest that the use of volume haptics with polygon-based graphics is a promising combination in terms of efficiency. That is one of the motivations for the comparison presented here. All haptics algorithms evaluated in this section, including voxel-based algorithms, are combined with polygonal-mesh graphics rendering for visualization.

4.3 Literature Review

In the past, different techniques have been implemented to provide force feedback with polygonal meshes, volumetric data, and intermediate representations. This section describes some approaches found in the literature.

4.3.1 Polygonal Mesh Haptics Rendering

Polygonal mesh methods require 3D models to be represented as rigid polyhedra obtained from the original dataset. Within these methods, an algorithm used for single-point contacts was proposed in (Zilles, Salisbury, 1995). This method used a "god-object" to constrain the haptic interface point to the mesh surface, avoiding penetration. The tip of the haptic device was coupled to the proxy through a spring model. In each haptic frame the force rendered was proportional to the distance between the probe and the proxy. A virtual proxy point of finite size, to avoid fall-through due to numerical gaps in polygonal meshes, was proposed in (Ruspini et al., 1997). This paper also proposed HL, a haptic interface library based on a graphics library (GL) from Silicon Graphics. The proxy method and the idea of a haptic library based on OpenGL were later implemented in SensAble's OpenHaptics (SensAble Technologies OpenHaptics; Itkowitz

et al., 2005). It offered two alternative haptics rendering modes: Feedback Buffer (based on OpenGL 3D polygonal primitives) and Depth Buffer (based on the OpenGL depth-buffer). Feedback Buffer delivered high quality collision detection and force feedback but the number of polygons it could handle was limited. On the other hand, Depth Buffer was relatively insensitive to the number of polygons because it was based on a 2D image drawn on the Z-buffer. However, it exhibited "noticeable [force] discontinuities when feeling shapes with deep, narrow grooves or tunnels" (SensAble OpenHaptics Toolkit Version 3.0 Programmer's Guide [a]).

In addition to point-based algorithms, there are also line-based approaches, such as (Basdogan et al., 1997). The authors presented a ray-based method to detect collisions between 3D polygonal objects and the haptic stylus, which was modeled as a line segment. When a collision was detected, the distance between the collision point and the tip of the stylus was computed, and the reaction force in the normal direction was made proportional to that distance using a simple spring-damper model. Static and dynamic frictional forces were also computed in the tangential direction.

As pointed out in (Basdogan et al., 2007) and (Lundin, 2007a), the fact that polygonal meshes were generated from isosurfaces prevented the user from dynamically modifying the model during the simulation, since it was computationally expensive to regenerate the whole mesh in real-time. Having this ability is, though, an essential requirement for modeling surgical procedures where volume removal is frequently required, e.g. bone drilling.

4.3.2 Volume Haptics Rendering

Iwata and Noma (1993) presented an approach called Volume Haptization to provide force feedback from volumetric datasets. For scalar data, they mapped either the voxel values to torque vectors or the gradient of voxel values to force vectors. Avila and Sobierajski (1996) described a gradient method where the normal and viscosity force components at a given point depended on the material density and the gradient magnitude at that point. The disadvantage of these methods is that they can produce instabilities or undesired vibrations, especially in regions containing sharp transitions, where the gradient magnitude and direction can vary abruptly.

Volume haptics has been systematically studied in a series of publications (Lundin et al., 2002) (Lundin et al., March 2005) (Lundin et al., Nov. 2005) (Lundin et al., 2006) (Lundin, 2007b) (Lundin et al., 2008). In (Lundin et al., 2002), a method to generate surface and viscosity haptic feedback from volumes, along with simulation of material properties was presented. The method evolved in (Lundin et al., March 2005), where haptic primitives, such as directed force, point, line, and plane were used as building blocks for their proxy-based method. Based on those haptic primitives, a number of haptic modes were constructed, i.e. viscosity mode, gradient force mode, vector follow mode, and surface and friction modes. The method was refined in (Lundin et al., Nov. 2005), where a numeric solver to compute the final forces was described. In (Lundin et al., 2006), the Volume Haptics ToolKit (VHTK) was presented and implemented as an extension to SenseGraphics H3D library (H3D). An analytical solver, which falls back into their numerical solver when its requirements are not satisfied, was introduced in

(Lundin, 2007b). Finally, a method which contemplates time-varying volumetric data was introduced in (Lundin et al., 2008).

4.3.3 Intermediate Representation methods

Intermediate representation methods were first proposed in (Adachi et al., 1996). The idea consisted of representing touchable surfaces at a given point by a virtual plane tangent to the surface at that point. The collision detection loop ran independently of the servoloop and was updated at a lower rate, whereas the servoloop was updated at a higher rate required to render stiff objects. Combining intermediate representations and lower update rates allowed to simplify the collision detection problem and to quickly detect collisions between the tip of the haptic device and the virtual plane. The method, however, had a fundamental limitation. If the update rate for the virtual plane (computed in the collision detection loop) was too low, the operator could perceive discontinuities as the proxy "jumped" from one plane to another. This problem was addressed in (Mark et al., 1996), where the recovery time method was presented. The method reduced the magnitude of the force immediately after a new virtual plane is computed, allowing to gradually and smoothly bring the tip of the haptic stylus to the new surface. A simple algorithm using the intermediate representation method on volumetric data was presented in (Chen et al., 2000). The algorithm extracted virtual planes from the volumetric data without the need of precomputing isosurfaces. This algorithm, combined with a proxy-based method, allowed generation of haptic feedback directly from the volumetric data. Similarly, (Rizzi et al., 2010) presented a collision detection algorithm (introduced in Chapter 3) based on determining where a line segment intersects an isosurface defined by transfer functions that depend on voxel intensities. A line segment was created from the

position of the haptic interaction point in the previous haptic frame and its current position. If a collision was detected, the point where it occured as well as the surface normal at that point was computed. With that information the underlying haptics library was able to compute feedback forces as if it was working with polygonal models. In (Körner et al., 1999), an intermediate local representation which uses Marching Cubes to generate isosurfaces from voxel data adjacent to the haptic stylus position was proposed. Local isosurfaces from a 7x7x7 cube were passed to the servo loop in the haptics library as an intermediate representation of the local volume data.

4.3.4 Evaluation of haptic algorithms

A number of evaluation methods for haptic algorithms have been proposed in the literature. The dependency of haptic algorithms on the user's input has been pointed out in (Ruffaldi et al., 2006). The same paper described a methodology for evaluating haptic algorithms based on recording actual forces and trajectories from a user interacting with a real object. The recordings were used as inputs to the algorithm being tested and compared with its output. Similarly, (Srimathveeravalli et al., 2009) presented a virtual handwriting simulator where the position of the haptic device and its forces were recorded. For validation purposes, the haptic device was coupled to a robotic arm programmed to imitate typical inputs from a human user. In this way, the forces generated by the haptic device in response to reproducible inputs were recorded and their variations analyzed.

4.4 Algorithms Evaluated

Multiple algorithms for haptic interaction with isosurface models are evaluated in this work. The selection of algorithms is based on the following criteria: (i) an implementation must be available using OpenHaptics, consequently making use of SensAble haptic devices; and (ii) the algorithm must provide haptic feedback from isosurfaces. Seven algorithms satisfying the requirements were identified, as listed in Table II.

TABLE II

HAPTIC ALGORITHMS EVALUATED

	Algorithm	**Rendering type**	**API**	**Nomenclature**
1	OpenHaptics' Feedback Buffer (Itkowitz et al., 2005)	Polygonal Mesh Rendering	H3D	FB
2	OpenHaptics' Depth Buffer (Itkowitz et al., 2005)	Polygonal Mesh Rendering	H3D	DB
3	VHTK's ScalarSurfaceFriction mode (Lundin et al., 2006)	Volume Haptics Rendering	H3D	VHTK
4	Intermediate representation algorithm in (Rizzi et al., 2010)	Intermediate Representation Methods	Immersive Touch	IR
5	God Object (Zilles, Salisbury, 1995)	Polygonal Mesh Rendering	H3D	GodObject
6	Ruspini method (Ruspini et al., 1997)	Polygonal Mesh Rendering	H3D	Ruspini
7	Chai3D (CHAI3D)	Polygonal Mesh Rendering	H3D	Chai3D

A fundamental idea in OpenHaptics HLAPI is to emulate the interface of OpenGL (Itkowitz et al., 2005), getting the geometry to be haptically rendered from graphics primitives. In Feedback Buffer, the first algorithm evaluated, OpenHaptics captures all the geometric primitives that generate points, lines and polygons from OpenGL. One of the limitations of this mode is that it is required a priori to tell the API the number of vertices to be rendered for buffer allocation. In case of Depth Buffer, the second algorithm, an image rendered on the OpenGL depth buffer by a haptic camera is used to simplify computations and reduce buffering requirements. Its disadvantage is that, when disabling the haptic camera view optimization provided by OpenHaptics, only part of the geometry visible from the viewpoint used to render the shapes are touchable. Even enabling the haptic camera view optimization, "*noticeable force discontinuities are felt when touching shapes with deep, narrow grooves and tunnels*" (OpenHaptics Toolkit Version 3.0 Programmer's Guide [b]).

The third algorithm evaluated, ScalarSurfaceFriction in VHTK (Lundin et al., 2006), uses a plane determined by the surface gradient to detect touchable surfaces from volumetric data. Transfer functions are used to specify the strength of the surface and its friction as a function of the voxel intensities. There is an additional parameter called distinctness which is based on the magnitude of the gradient. This mode can be used from the H3D environment using X3D and Python scripts.

The fourth algorithm (Rizzi et al., 2010), while based on voxels, preserves desirable features from polygonal mesh models. Taking advantage of OpenHaptics' custom shapes, it creates isosurfaces on-the-fly from volumetric data which OpenHaptics uses exactly as it does with polygonal meshes. This also means that models can use stiffness, damping,

dynamic and static friction parameters as well as event callback functions (touch, untouch, motion) available to polygonal models in OpenHaptics. This algorithm runs in the OpenHaptics servo loop.

GodObject (Zilles, Salisbury, 1995) and Ruspini (Ruspini et al., 1997) renderers are implemented as part of HAPI, a low-level layer in the (H3D) API. H3D also includes an option to render polygonal models using the (CHAI3D) library. Due to their availability, these implementations in H3D are the ones used in this work.

4.5 Experiments and results

In our experiments, the running time for each servoloop frame is measured to determine if a given algorithm is able to maintain the required rate of 1 KHz. In addition, haptics quality of the algorithms is evaluated based on the continuity of forces generated when performing a specific task. Finally, the performance of all algorithms is evaluated in terms of rendering time in the client application thread.

4.5.1 Description of the Experiments

The common part to all our experiments consists of haptically exploring anatomical models, maintaining contact with the smooth surface (skull shown in Figure 10) at all times. In the interest of generating reproducible inputs to the algorithms, we considered the approaches described in (Ruffaldi et al., 2006) and (Srimathveeravalli et al., 2009). Using pre-recorded trajectories and injecting them into the algorithms (Ruffaldi et al., 2006) was not possible, as OpenHaptics receives its input from the haptic device. Similarly, using another device coupled to the haptic device to provide its input

(Srimathveeravalli et al., 2009) is not applicable, as we need to capture and evaluate the intrinsic variability of human users interacting with the models in a closed loop.

Our solution consists of having the user follow a pre-recorded trajectory that has been converted to animated VRML and X3D files. In the experiments, a red sphere traverses the pre-recorded path continuously and at constant speed (Figure 10). The red sphere is animated and moves following a straight-line trajectory on the 3D surface from the green sphere (starting point) to the blue (end point) at constant velocity. The operator is instructed to follow the red sphere while trying to maintain contact with the surface at all times, starting from the green sphere and ending in the blue one. This simple arrangement guarantees repeatability without the need to introduce haptic constraints to the user movement, which could affect the force rendered by the algorithms.

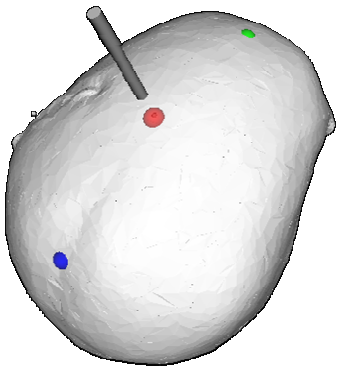


Figure 10. Predefined trajectory for the experiments.

4.5.2 Experiment 1 - Servoloop Frame Rate

It is well known that a minimum frame rate of 1 KHz is required to haptically display moderately stiff objects. The objective of this first experiment is assessing the performance of the algorithms by measuring their servoloop frame rate.

For haptic algorithms based on polygonal meshes, the average servoloop execution time is computed and presented in Figure 11 as a function of the number of polygons in the model. DepthBuffer and FeedbackBuffer remain insensitive to the number of polygons and their average frame rendering times are well below 1 msec. A dependence on the number of polygons is clearly visible for Chai3D, Ruspini and GodObject methods. The dependence is not significant for the GodObject algorithm, whereas it is more pronounced for the Ruspini and Chai3D methods. Furthermore, for models of approximately 185K polygons and up, Chai3D average frame rendering times are above 1 msec (represented by the bold red line in Figure 11), which means Chai3D is not able to maintain a haptics frame rate of 1 KHz for moderately complex polygonal models.

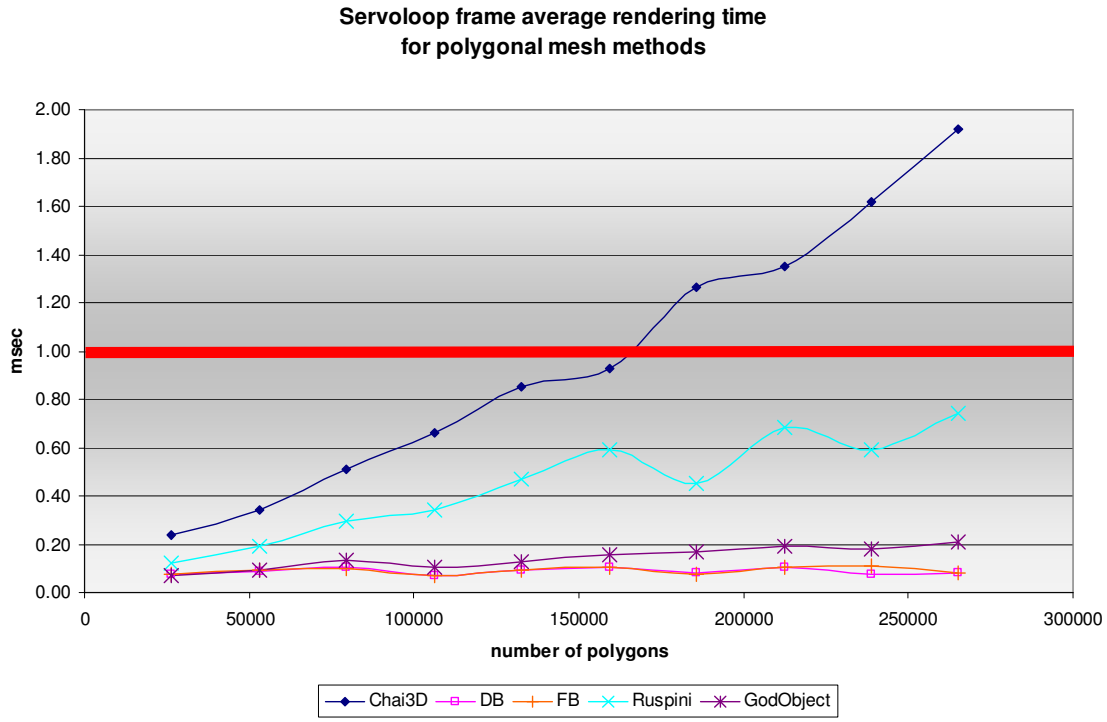


Figure 11. Servoloop average rendering time for polygonal-mesh methods.

Volume-based haptic algorithms are independent of the number of polygons being visualized; hence in this experiment the results for VHTK and IR are not presented as a function of the number of polygons. Instead, a normalized histogram of the sampled servoloop frame rendering time is shown in Figure 13. For the volumetric model described in section 3.5.1, VHTK's peak is at 0.08 msec, while IR's peak is at 0.14 msec. The distributions for both methods decay to zero well below the critical value of 1 msec, shown also as a bold red line at the rightmost side of Figure 13.

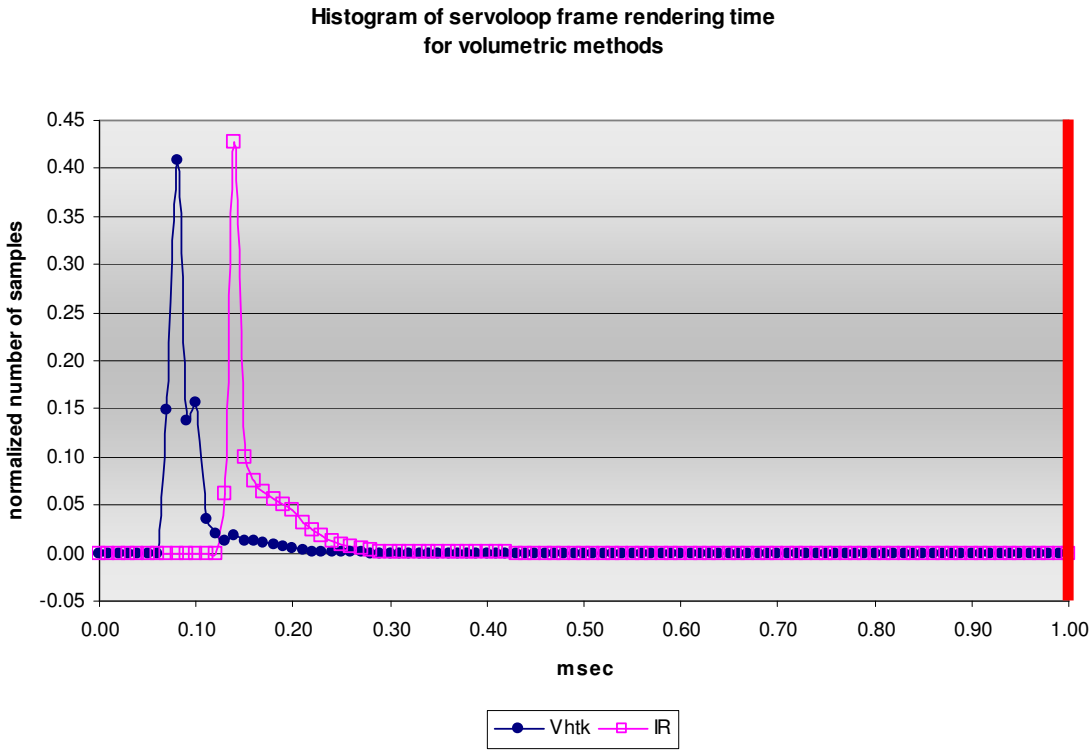


Figure 12. Servoloop frame rendering time for volumetric methods.

4.5.3 Experiment 2 - Force Rendering

Evaluation of force rendering is essential to determine the quality of force feedback provided by a haptic algorithm. Force discontinuities and fall-through, where a collision

is not properly detected, are among the major causes of improper force feedback perceived by the user. In order to quantify the quality of haptic feedback for each algorithm we have defined a metric called *force anomaly coefficient*.

4.5.3.1 Force anomalies

Even though the volumetric dataset and its related polygonal models used in the experiments are relatively smooth, they pose a challenge to the algorithms under study. Some of the anomalies observed in the experiments include fall-through, force irregularities and discontinuities, as well as the haptic device getting stuck into the surface.

We are interested in quantifying the smoothness of forces rendered, and therefore, the derivative of the recorded forces is used for this purpose. In our method, the derivative of the force magnitude is computed for each case using the central differences method. Once a vector containing force derivatives is calculated, the average force magnitude and its standard deviation are computed. A force anomaly is detected for each element of the force derivative vector whose value is outside the interval $[\mu - 10 * \sigma, \mu + 10 * \sigma]$, where μ is the average of the force derivative vector and σ its standard deviation. The number of values lying outside the interval divided by the total number of force measurements for each case is what we call *force anomaly coefficient*.

It is important to mention that we have designed this metric specifically for smooth surfaces, such as the test case shown in Figure 11. For other less frequent cases in surgical simulation where non-smooth surfaces are required, this metric would not be entirely appropriate as in those cases large derivatives in the force magnitude are

expected. For the most frequent cases, though, we have found this metric adequately captures force discontinuities and fall-through occurrences.

The following table summarizes the process of obtaining the *force anomaly coefficient*:

TABLE III

OBTAINING THE FORCE ANOMALY COEFFICIENT

1) Given a vector F containing recorded forces, compute |F| and calculate its derivative using the central differences method. Store the result in vector |F|'
2) Compute the mean and standard deviation of vector |F|'
3) Find the values of |F|' outside the interval $[\mu - 10 * \sigma, \mu + 10 * \sigma]$. That is the number of force anomalies
4) Normalize the number of force anomalies by the total number of samples recorded

Figures 13-16 exemplify force anomalies detected in different experiments. Z components of the recorded trajectory are shown in blue whereas the detected anomalies are shown in red stars. The example shown in Figure 13 corresponds to Chai3D, where the haptic stylus got stuck on the surface near the end of the trajectory (right of the figure).

Although there is a deviation from the prescribed trajectory, there is no actual fall-through in Figure 13. The irregularities in the trajectory correspond to the operator pulling the haptic stylus trying to release it from the stuck position. The system reacts with high-magnitude forces as the operator tries to pull out the stylus and return to the predefined path. The peaks in red show anomalies detected in that particular region, where a strong force discontinuity was sensed.

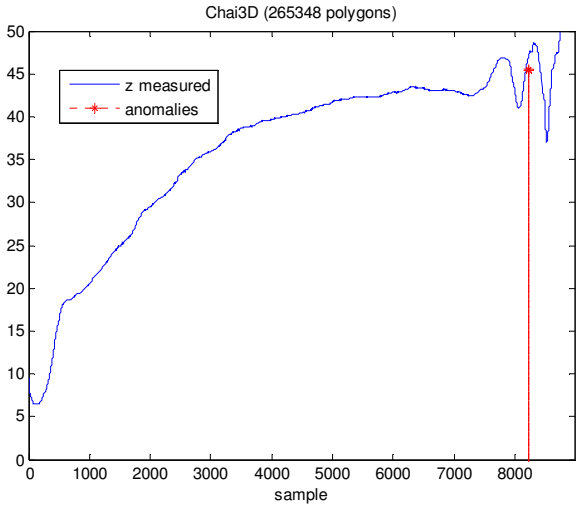


Figure 13. Force anomalies in Chai3D for 265K visualized polygons.

On the other hand, Figure 14 shows an actual occurrence of fall-through using the GodObject renderer in H3D, for a model consisting of 159K polygons. Force discontinuities are detected in the zone where fall-through occurs, as well as in another two regions where minor deviations from the ideal trajectory are found. Both previously discussed examples show that our evaluation method is able to identify deviations from the ideal path.

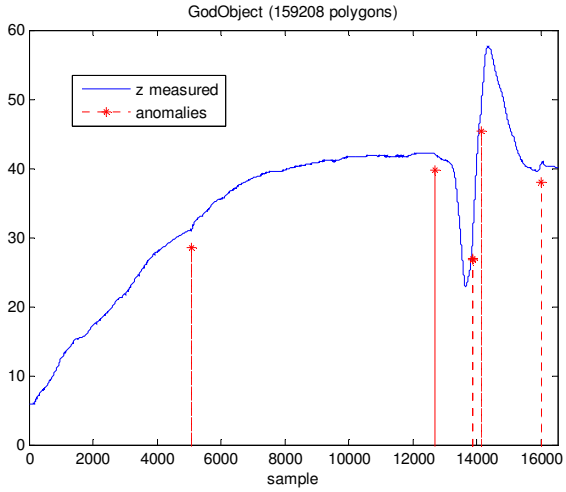


Figure 14. Force anomalies in GodObject for 159K visualized polygons.

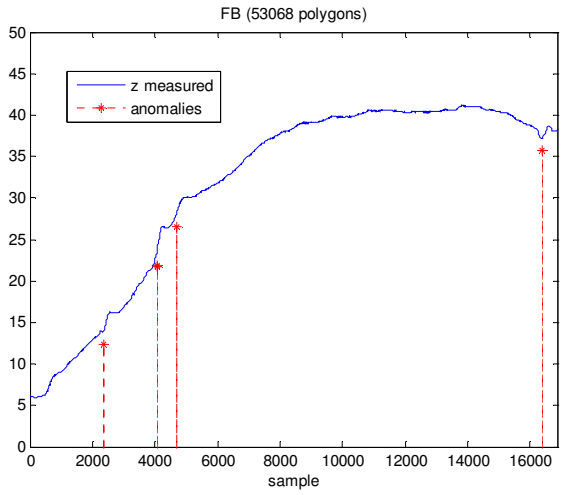


Figure 15. Force anomalies in FeedbackBuffer for 53K visualized polygons.

The method also detects irregularities of rendered forces. Figure 15 presents data for the OpenHaptics renderer in H3D using FeedbackBuffer, for a model consisting of 53K polygons. The operator felt certain spots along the trajectory with sudden high-level friction which are visible as small deviations from the trajectory and detected as force anomalies.

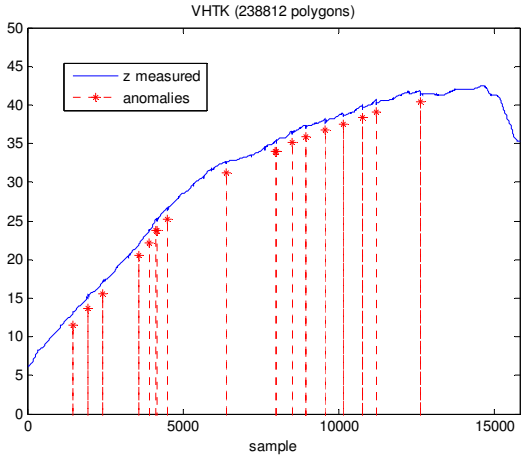


Figure 16. Force anomalies in VHTK for 238K visualized polygons.

In a similar way, an experiment using VHTK is shown in Figure 16, combined with surface graphics rendering of a model consisting of 238K polygons. Even though the friction transfer function was set to zero, the operator felt a rough surface, which is shown by small peaks in the trajectory and detected by large force variations.

From the examples shown it is clear that the *force anomaly coefficient* not only detects anomalies in force rendering, but also it is able to detect deviations from the trajectory through their associated force discontinuities. Thus, the method provides a good metric for assessing haptics quality.

4.5.3.2 Force anomaly coefficient for experimental data

In the previous sub-section some specific cases were shown. Here we present results obtained from the whole data collected in the experiment. Figure 17 shows the average *force anomaly coefficient* for each algorithm analyzed, as well as their maximum and minimum values.

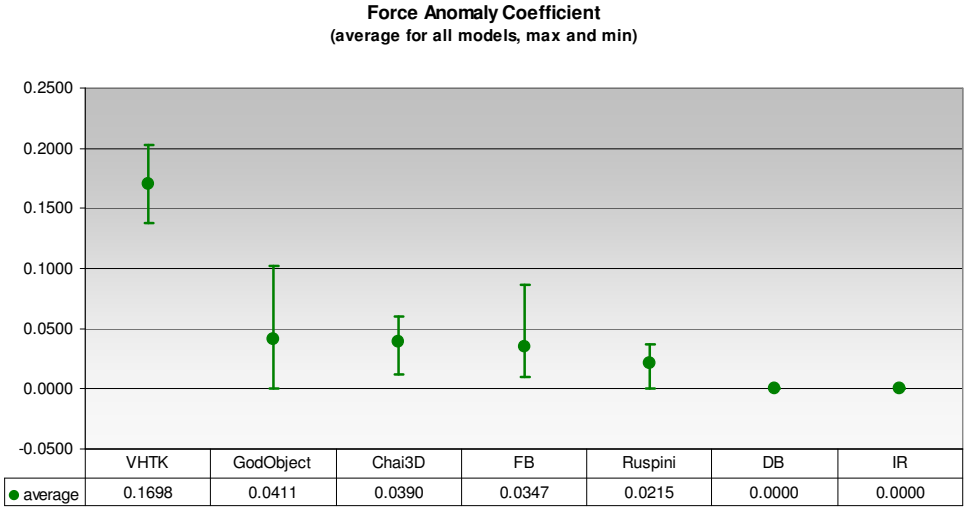


Figure 17. Average force anomaly coefficient for all cases and its range of variation.

It can be seen in Figure 17 that DepthBuffer does not show any force discontinuity or fall-through, in agreement with what the operator experienced. IR, the volume haptics algorithm implemented in the ImmersiveTouch API also produced an absolutely smooth force feedback in all cases. All other haptics rendering algorithms presented anomalies at least once during the experiment, shown by non-zero values of their *force anomaly coefficient*.

4.5.4 Experiment 3 - Client thread running time

Measuring the running time of the haptics rendering code in the client thread gives us additional insight for evaluating the algorithms from a practical point of view. As shown in Section 4.5.2, almost all algorithms are able to maintain the required 1 KHz frame rate in the servo thread. However, there are substantial differences when measuring the running time in the client thread, which is executed at graphics rendering rates (30-60 Hz). This metric directly determines the usability of each algorithm, since it affects the effective interaction frame rate of the final application as perceived by the user.

Figure 18 shows the average running time of each algorithm as a function of the number of polygons in the models. It is important to remark that the number of polygons used as independent variable is common to all algorithms (even voxel-based haptic methods), as it refers to the number of polygons used for graphics visualization.

For polygonal mesh methods, the independent variable is also the number of polygons used for haptics, whereas for voxel-based haptics the model remains invariable (only the associated polygonal graphics models change). Since we are comparing the combined graphics and haptics execution times, it is still fair to compare polygon-based and volume-based haptic algorithms in a single experiment and show the results as a

function of the number of polygons visualized. If a rigorous comparison among all methods is desired, one should only account for the last set of measurements in Figure 18 (rightmost side). In that case, the polygonal mesh representing the model is directly comparable to their voxel-based counterpart, as the polygonal mesh has been obtained by the Marching Cubes from voxels and no decimation has been applied. All other points in Figure 18 are presented to show dependence on polygon count for polygonal mesh-based methods.

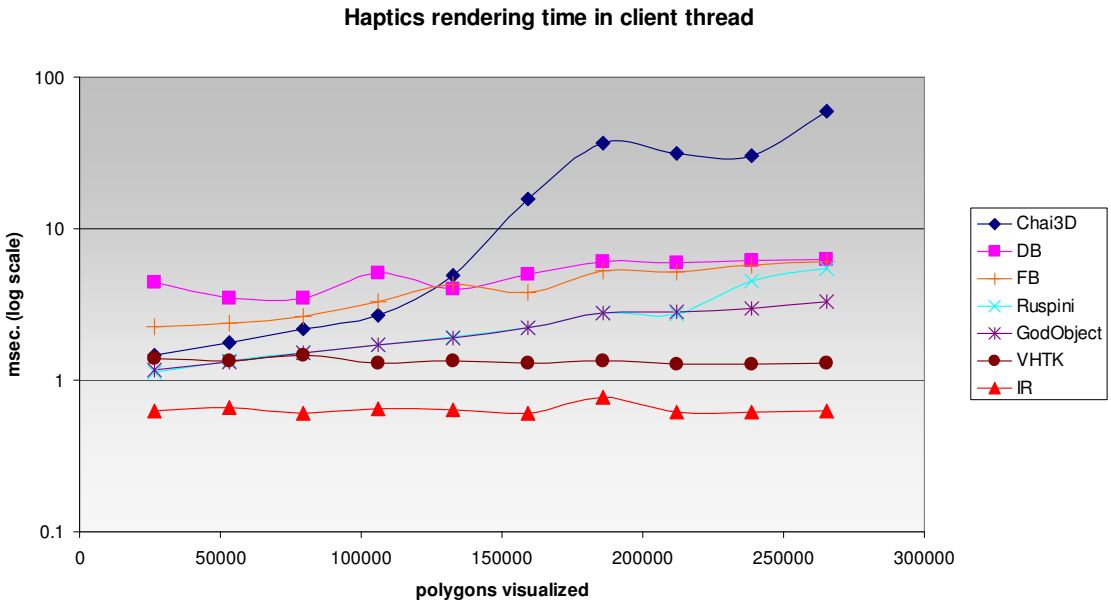


Figure 18. Run time for haptics rendering in client thread.

As expected, haptics algorithms based on polygons demand more time to process the geometry as the number of polygons increase. It is also important to highlight that both voxel-based approaches (VHTK, IR) are insensitive to the number of polygons visualized, since once the volume model is loaded in memory it is not necessary to regularly update model geometry in the client thread. Therefore, their client thread execution time for haptics updates remains essentially invariant.

Since graphics and haptics rendering in the main application thread are intimately related, we also measured the graphics rendering time for each case. As expected, the overall application frame rate is directly dependent on the combined haptics and graphics rendering time. Figure 19 shows the average combined haptics and graphics running time.

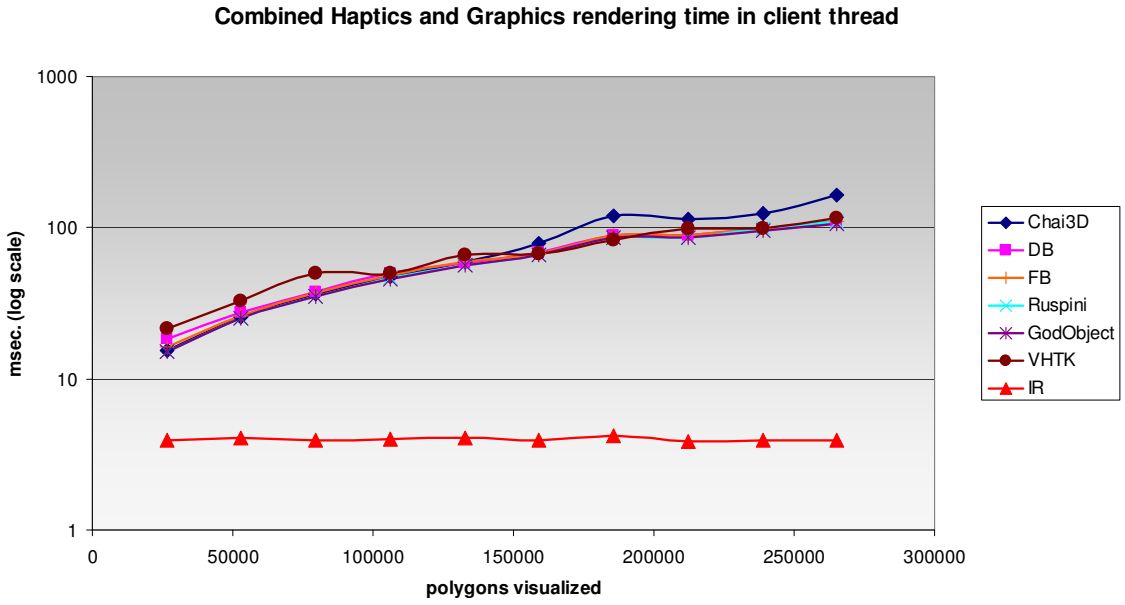


Figure 19. Combined run time for graphics and haptics in client thread.

Note that all algorithms implemented in H3D have the same graphics renderer, which has a significant impact on the overall performance. Our graphics rendering took an average of 3.5 msec independent of the number of polygons, the reason being a number of optimizations (Vertex Buffer Objects or VBOs) present in Coin3D. In contrast, the graphics rendering time in the H3D API increases significantly with the number of polygons.

As a result, it can be seen that all combinations of haptics and graphics increase their combined execution time with the number of polygons, except for our algorithm

(IR) combined with Coin3D, which maintains an almost constant execution time with 3D models of up to 270K polygons.

4.6 Contributions

Based on the results reported above, the problems observed for each algorithm are presented in Table IV. The algorithms have been arranged from top to bottom according to their perceived usability (increasing downwards).

TABLE IV

OBSERVED ISSUES

Chai3D	Unacceptable servoloop frame rendering times combined with the largest rendering time observed in client thread.
Ruspini	Servoloop frame rendering time markedly dependent on the number of polygons and expected to become unacceptable for more than 400K polygons. Moderate force anomaly coefficient.
VHTK	Largest number of force anomalies.
Depth/Feedback Buffer	Other than Chai3D, highest client thread rendering time. Larger number of force anomalies in Feedback Buffer and higher client thread rendering time in Depth Buffer.
GodObject	Moderate number of force anomalies, lower client thread rendering time.
IR	No issues observed.

Our novel haptics algorithm combined with polygonal mesh visualization appears to be the most efficient method for medical simulation using highly complex models. Among all the evaluated methods it provides the lowest total rendering time, it is insensitive to model complexity, and it correctly generates haptic feedback in all cases.

.

5. A HAPTICS ALGORITHM FOR MULTIPOINT COLLISION DETECTION

One of the major limitations of point-based haptics algorithms, such as the ones evaluated in the previous chapter, is that they can not simulate complex interactions between virtual surgical instruments and virtual 3D models of patient anatomies. These algorithms are only able to detect point-to-object collisions. This means that only the tip point of the virtual instrument is considered for computing its interaction with other virtual objects. This is a serious limitation when complex procedures involving object-to-object interaction are required, as it is the case when all points in the surface of the virtual instrument may collide with the simulated human anatomy, not only its tip.

The ventriculostomy module in (Luciano et al., 2006) used GHOST library and was later ported to OpenHaptics, therefore only the tip of the virtual catheter was allowed to interact with the virtual skin, skull, brain, or ventricles. If the surgeon moved the catheter laterally once it had been inserted into the brain, it was not possible to feel the walls of the burr-hole in the skull, drastically reducing the realism of the simulation. These undesired effects were compensated using haptic effects (i.e. the instrument was constrained to move in a straight line once the brain had been penetrated), yet a multipoint collision detection algorithm would be a more robust solution.

5.1 Research Problem

Extend the volume haptics algorithm of Chapter 3 to detect object-to-object collisions

5.2 Overview

The voxel-based haptics algorithm in Chapter 3 can be extended to account for multipoint collision detection. In its original single point implementation, a unique point, usually located in the tip of the instrument, is evaluated for collisions with isosurfaces representing virtual anatomies. The original problem geometry from Chapter 3 is repeated in Figure 20 for convenience.

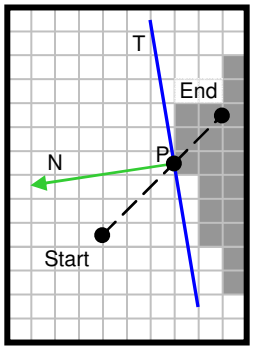


Figure 20. Problem geometry in single point collision detection algorithm.

A simple way of extending this algorithm for multipoint collision detection consists of defining the virtual instrument by multiple points, akin to the PointShells in the Voxmap algorithm (McNeely et al., 1999). Figure 21 shows a virtual craniotome and multiple points (in red) defining its effective contour for multipoint collision detection.

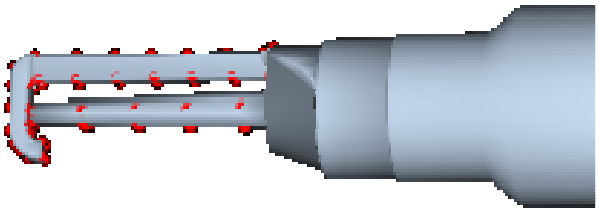


Figure 21. Craniotome with multiple points for collision detection.

As in the original algorithm in Chapter 3, the position and orientation of the virtual instrument is known in the current and previous haptics frames, corresponding to the End and Start points in figure 20, respectively. Similarly, the Start and End positions for every point representing the virtual tool are also known in the current and previous haptics frame. Therefore, by evaluating a line for each point and detecting their intersection points with the isosurface, the initial algorithm can be extended to detect collisions using multiple points.

The volume haptics algorithm can be extended in such a way that multiple lines are evaluated simultaneously. This allows us to determine multiple potential contact points, with which a resulting force, and the point where it should be applied, can be computed. A single force and its point of application are sufficient to provide 3-degree-of-freedom haptic feedback; therefore it should be possible to effectively extend OpenHaptics and our volume haptics algorithm to detect object-to-object collisions.

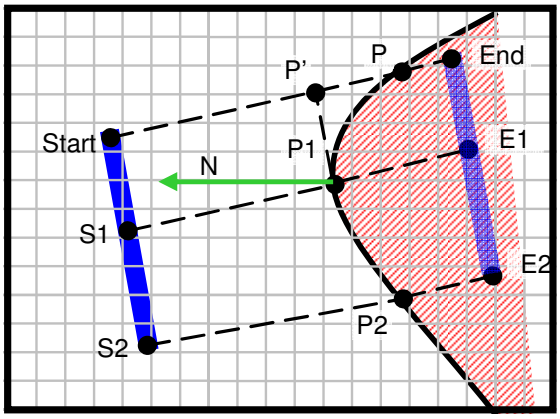


Figure 22. Multipoint problem geometry.

5.3 Algorithm Details

The new problem geometry is shown in Figure 22 (compare with Figure 20). The virtual instrument is represented in a simplified form by a blue bar. Also for simplicity, the virtual instrument contains three points; Start is the tip of the instrument, and actually the only point that OpenHaptics keeps track of. S1 and S2 are additional points along the main axis of the virtual instrument that will be used for collision detection. In this way, when the virtual instrument moves in space to the position shown in light blue, colliding in its way with the object, the new positions for the reference points will be End, E1 and E2. Correspondingly, the intersection points with the isosurface will be P, P1 and P2. In order for the collision to be detected properly, P1 (the closest intersection point to the original position of the instrument) should be the intersection point with the isosurface. However, the haptics library expects to receive a point along the line determined by the Start and End points. Therefore, the point P' (the perpendicular projection of P1 to the line defined by Start and End) is returned, together with the normal vector N at P1. In such a way, the haptics library will adjust its proxy position to prevent fall-through as if a single point collision had been detected in P'. As a result, the user will effectively perceive the virtual instrument as if it were made of multiple points that can interact with the virtual models.

5.4 Implementation

For a virtual instrument represented by N points, the first alternative to implement the algorithm presented in the previous section is to traverse the N parallel lines simultaneously and stop the iterations as soon as the first colliding point is detected (P1

in Figure 22). Figure 23 presents the algorithm as nested loops, where the outer loop moves forward along the individual lines and the inner loop evaluates points in each parallel line for a given step of the outer loop (compare with algorithm in Figure 8)

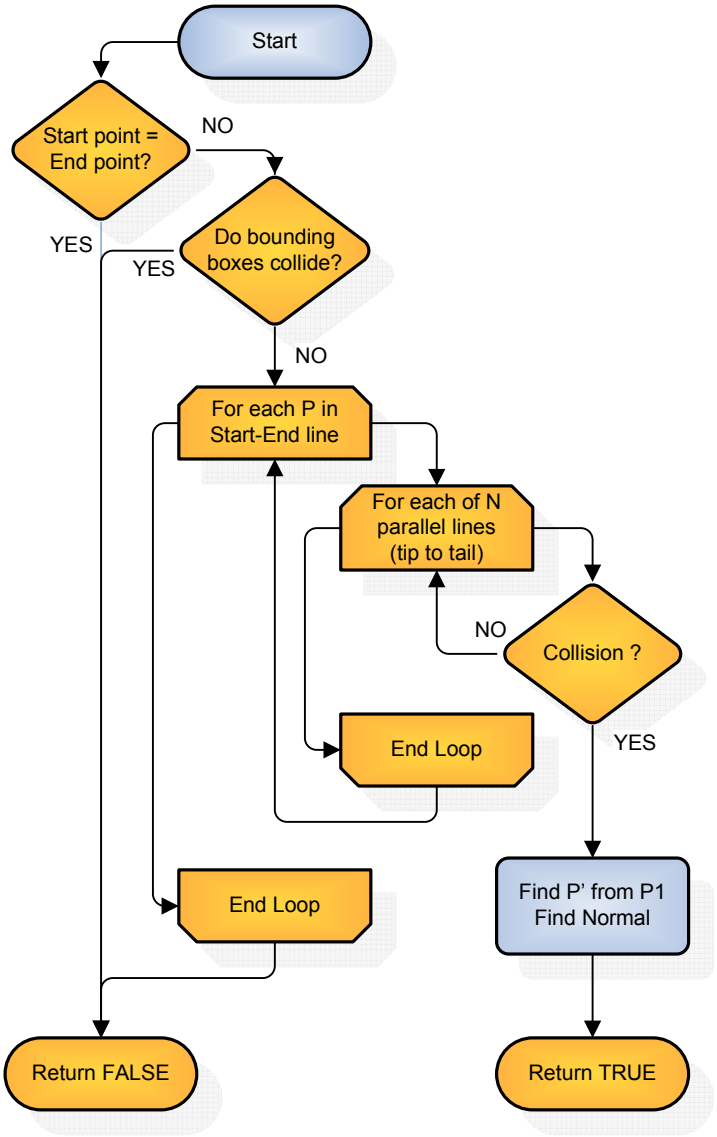


Figure 23. Multipoint collision detection algorithm.

Even though this implementation is straightforward, it suffers from a non-obvious minor flaw. When the virtual instrument is being moved tangentially to the isosurface, as it is usually the case when navigating the contour of spinal pedicles, the direction in

which the points composing the virtual instruments are evaluated for collisions may play an important role in preventing fall-through effects.

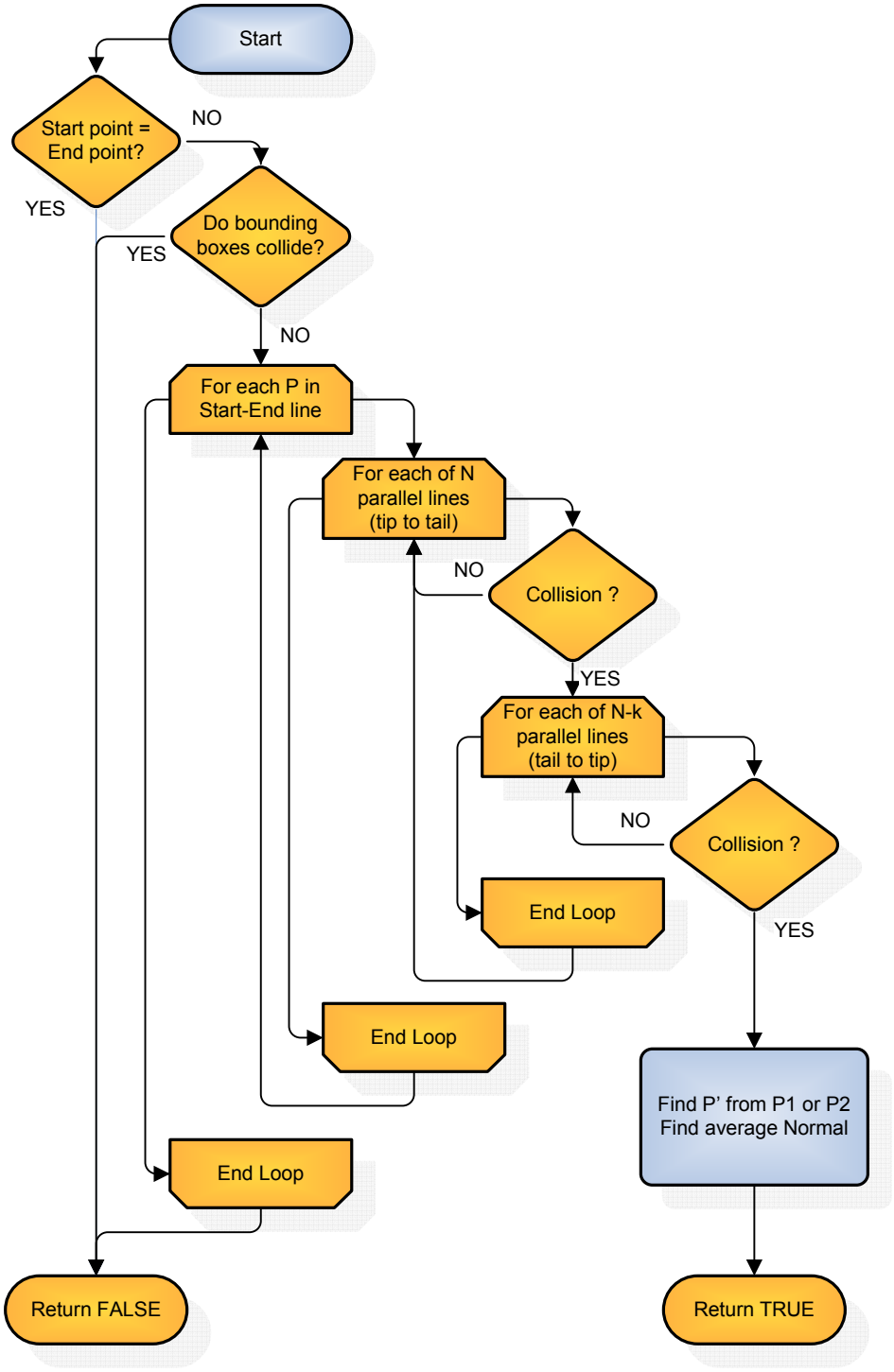


Figure 24. Refined algorithm for multipoint collision detection.

In the algorithm implementation of Figure 23 the virtual instruments points are traversed from tip to tail. This is not a problem when moving along the isosurface tangentially, as long as the virtual instrument moves in the forward direction. As soon as it starts moving backwards, it would be preferable to evaluate the N points in the virtual instrument from tail to tip, to avoid potential fall-throughs.

The problem is avoided by traversing the points in both directions and detecting two simultaneous collision points. Figure 24 shows a refinement of the algorithm in Figure 23 where there are two searches for colliding points within a given step without increasing the algorithm complexity. One of the loops searches for P1, the first colliding point from tip to tail, whereas the other loop traverses the list of points from tail to tip to detect P2, eventually stopping at *k* (the point detected by the tip to tail search). The actual contact point P' returned by the algorithm is the same for either P1 or P2 (since they both are at the same distance from the virtual tool), whereas the normal vector returned is the average from the normals found at P1 and P2.

5.5 Limitations

The original volume haptics algorithm was designed to provide 3-degree-of-freedom force feedback, with the inherent drawback that it can not provide torque feedback. Consequently, the extension in this chapter is unable to provide torque feedback. Without torque, it is not possible to prevent the virtual instrument from penetrating the 3D models in some specific cases. For example, when the virtual instrument undergoes a pure rotation (i.e. the body rotates around an axis passing through the tip, but the tip is not displaced) the multipoint algorithm detects the collision but the

resulting force applied in the tip of the instrument is not enough to prevent fall-through. This particular scenario is illustrated in Figure 25, where the virtual instrument is under pure rotation, as shown by the red arrow. In these conditions, and without a torque-enabled haptic device, it is not possible to prevent undesired fall-through into the interior of the virtual vertebral body. However, a workaround to this limitation is presented in the following section.

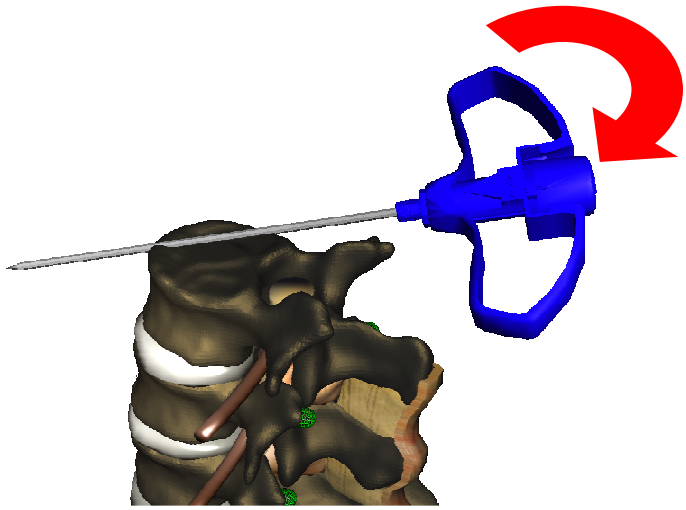


Figure 25. Pure torque problem.

5.5.1 Locking the cursor to help prevent fall-through

In previous simulation modules, haptics effects have been extensively used to help alleviate undesired effects of point-based haptics rendering. For example, line-locking effects were activated when anatomies like brain or bone were penetrated by the virtual instrument to compensate for the lack of side-wall collision detection. Similarly, to help prevent the pure-torque problem in Figure 25, a cursor locking mechanism was implemented. The effect consists of detecting a pure rotation along with a collision between the virtual instrument and virtual anatomies that could potentially cause an

undesired fall-through. When those conditions are detected, the cursor tracking is disabled, effectively locking its visual representation in the 3D workspace and freezing the position of the N points used for collision detection, thus preventing the potential fall-through.

5.5.2 Conditions to lock the cursor

Traversing the points in the virtual instrument from both ends, as in the algorithm shown in Figure 24, may result in two different points (*h* and *k*) that collide with the isosurface. If these points are detected to be at a distance larger than a predefined constant D (in millimeters), it signals that the undesired effect shown in Figure 25 is starting to occur. In that moment, it is necessary to activate the locking mechanism to prevent the fall-through to happen. Therefore, the conditions to lock the cursor are:

$$\begin{cases} h & \text{collision detected (tip to tail)} \\ k & \text{collision detected (tail to tip)} \\ |h - k| > D & \end{cases}$$

When the conditions to lock the cursor are satisfied, it is also necessary to establish the geometric conditions under which the cursor will be unlocked. For that purpose, a normal vector must exist at point h. The current dot product between the normal at h and the orientation of the virtual instrument is stored in dotProd and will be used to check if the cursor can be unlocked.

TABLE V

ALGORITHM TO LOCK CURSOR ORIENTATION

ALGORITHM:	**Lock cursor orientation**	
INPUT :	point h	collision point from tip to tail
	point k	collision point from tail to tip
	distance D	distance constraint in millimeters
	boolean Lock	FALSE means that cursor is not locked
OUTPUT :	float dotProd	dot product when cursor was last locked
	vector Normal	normal vector at h
	boolean Lock	TRUE means that cursor must be locked

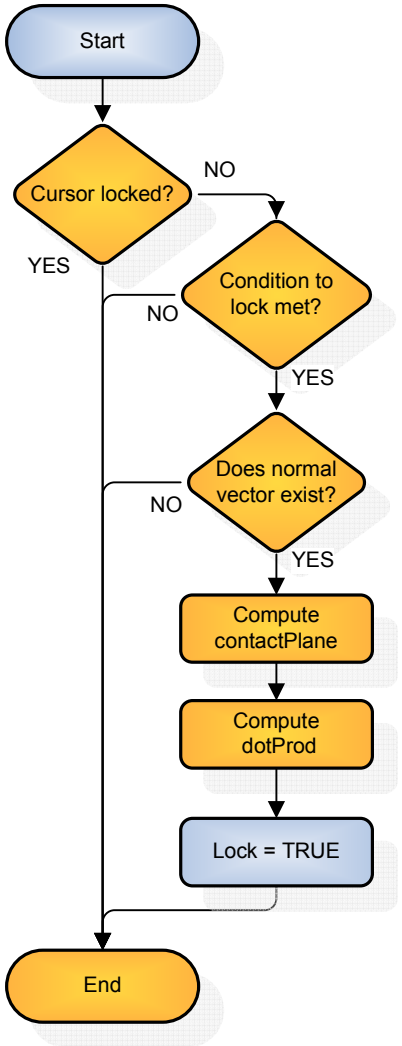


Figure 26. Algorithm to lock cursor.

5.5.3 Conditions to unlock the cursor

When the cursor is locked (Lock variable is TRUE), the conditions to unlock the cursor must be checked. In other words, it is necessary to check the updated orientation of the virtual instrument and compare it with its orientation when it was locked. This is easily accomplished using the properties of the vector dot product. If the result of the current dot product between the normal vector at the instant the cursor was locked and the current orientation of the virtual instrument is larger than dotProd (the value of the same dot product when the cursor was locked), that means that the virtual instrument has been rotated in the opposite direction that caused the locking, and therefore it can be unlocked.

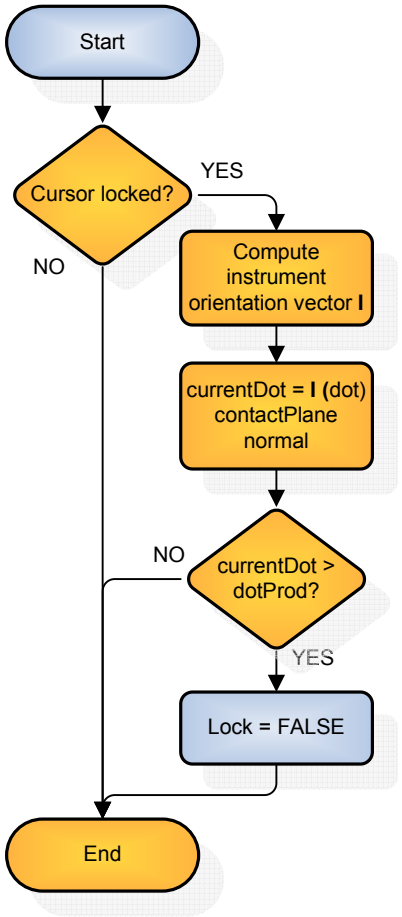


Figure 27. Algorithm to unlock cursor.

TABLE VI

ALGORITHM TO UNLOCK CURSOR ORIENTATION

ALGORITHM:	**Unlock cursor orientation**	
INPUT :	`float dotProd`	`dot product when cursor was last locked`
	`vector Normal`	`normal vector when cursor was last locked`
	`boolean Lock`	`TRUE means that cursor is locked`
OUTPUT :	`boolean Lock`	`FALSE means that cursor must be unlocked`

5.6 Contributions

The multipoint extensions to the original algorithm presented in this chapter allow us to simulate procedures in which the realistic modeling of the interactions between surgical tools and complex anatomies is of utmost importance to the surgeon. Existing modules to simulate central line needle placement and Jamshidi needle insertion for pedicle screws were modified with these extensions to provide multiple point collision detection. Examples and detailed descriptions of these modules are described in Chapter 8.

6. VOLUME HAPTICS AND POLYGONAL GRAPHICS FOR SIMULATION OF BONE REMOVAL PROCEDURES

Volumetric datasets have become essential in Virtual Reality (VR) and Haptics simulation for medical and surgical training. Technologies such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) provide 3D scans of patient anatomy, from which 3D models can be generated to represent highly detailed and complex anatomical structures such as bone, organs or muscle.

Combinations of scene graph managers and haptics libraries have been extensively used in surgical simulations for graphics and haptics rendering of 3D models. The goals of these simulations are simultaneous visualization of complex 3D models and tactile interaction with anatomy at interactive frame rates. While frame rates in the order of 30-60 Hz are acceptable for graphics rendering, the minimum required rate of 1 KHz for haptics rendering makes it a non-trivial problem when dealing with complex and highly detailed polygonal models. On the other hand, volume-based haptics techniques have proven themselves capable of generating force feedback from complex anatomies at interactive frame rates (Lundin, 2007; Rizzi et al., 2010).

The possibilities of our volume haptics algorithm in combination with a fast mesh regeneration algorithm for graphics are investigated in this chapter.

6.1 Previous work

There are significant works focusing on bone surgery using volume haptics. Gibson (1995) developed a prototype for haptic exploration of a 3D model of a human hip based on voxels. Using occupancy maps, collisions were easily detected and haptic feedback

was generated to prevent penetration of virtual models. However, this method requires the generation of occupancy maps, which is essentially a form of segmentation, demanding some preprocessing work. Petersik et al. (2002) presented a haptics rendering algorithm based on a multi-point collision detection approach. A ray-based algorithm was used for graphics and haptics rendering. While static objects were represented by voxels, the location of their surfaces was obtained by a ray-casting algorithm at sub-voxel resolution. This approach was found to be limited in the effective stiffness of the simulations (Morris et al., 2006). Also, this work presented a hybrid approach based on voxels for haptics rendering and on polygonal meshes for graphics rendering. As portions of the virtual bone are being removed, their algorithm modified the surface locally and then recomputed the meshes in real time, solving the stiffness problem in (Petersik et al., 2002). The remeshing process was performed locally, demanding a substantial increase of complexity in their algorithms which also limited its usability to non-complex models.

6.2 Research Problem

Combine the volume haptics algorithm with real-time graphics polygonal surface regeneration to efficiently simulate burr-hole drilling and other bone removal operations

6.3 A real-time algorithm for graphics polygonal surface regeneration

The Marching Cubes algorithm (Lorensen, Cline, 1987) is a well-known solution to the problem of creating an isosurface from volumetric data. Obtaining an isosurface is a simple way of constructing a polygonal mesh model from a volumetric dataset, such as a CT scan, where the isosurface value represents the intensity of the anatomy to be

extracted. In other words, Marching Cubes can be used to extract anatomies of interest from CT or MRI scans. The problem is that existing implementations of Marching Cubes are not suitable for real-time interaction.

In 2007, Nvidia introduced its Compute Unified Device Architecture (CUDA) for parallel computing using their Graphics Processing Units (GPU). As part of the CUDA Software Development Kit (SDK), a parallel implementation of the Marching Cubes algorithm was offered. A simplified description of the essential tasks to parallelize Marching Cubes for CUDA follows:

1. *Determine which voxels are going to contribute in the generation of polygons. Discard non-contributing voxels*
2. *Generate triangles using one GPU thread per voxel. Each thread will access information from neighbor voxels and create three vertices of a triangle and a normal vector. Triangle information is written in GPU global memory*
3. *Render geometry from previous step using special functions to interact with OpenGL provided by CUDA*

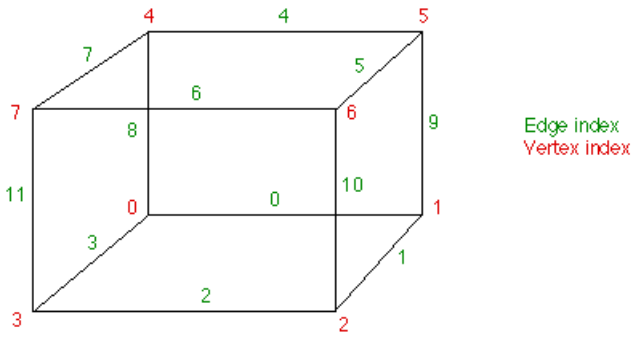


Figure 28. Indexing convention for vertices and edges (Bourke, 1994).

An efficient implementation of the algorithm has been described in (Bourke, 1994). Considering that a cube has eight vertices and twelve edges, where vertices take the values of their corresponding voxel intensities, we can index them as in Figure 28

Following the example in (Bourke, 1994), if the intensity value of the voxel in vertex 3 is below the chosen isosurface value and the other voxels are all above the isosurface value, then Marching Cubes would create a triangle intersecting edges 2, 3, and 11.

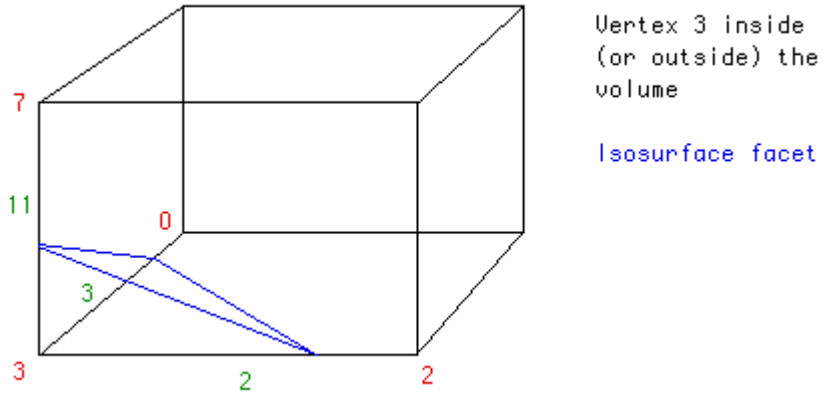


Figure 29. Example of triangle created by Marching Cubes (Bourke, 1994).

There are $2^8 = 256$ possible combinations of voxel intensities above or below the desired isosurface value in a given marching cube. Therefore, there are 256 possible configurations of triangles to be generated for every cube evaluated. By symmetry considerations, these 256 cases can be reduced to 14 cases, as illustrated in Figure 30:

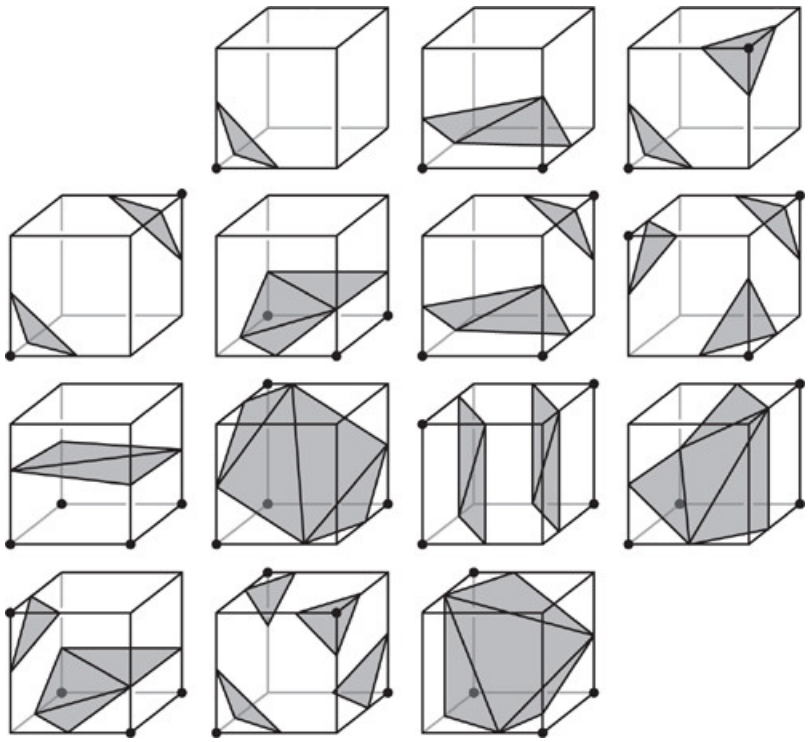


Figure 30. Marching Cubes fundamental cases (Geiss, 2007).

However, based on performance considerations, the reference implementation uses a lookup table taking into account the 256 possible cases. In such a way, an 8-bit index is computed, where each bit corresponds to a given voxel (Bourke, 1994):

```
cubeindex = 0;
if (voxel[0] < isolevel) cubeindex |= 1;
if (voxel[1] < isolevel) cubeindex |= 2;
if (voxel[2] < isolevel) cubeindex |= 4;
if (voxel[3] < isolevel) cubeindex |= 8;
if (voxel[4] < isolevel) cubeindex |= 16;
if (voxel[5] < isolevel) cubeindex |= 32;
if (voxel[6] < isolevel) cubeindex |= 64;
if (voxel[7] < isolevel) cubeindex |= 128;
```

There are two lookup tables, edgeTable and triTable indexed by the value of `cubeindex`; edgeTable contains 256 elements of 12 bits, where each bit corresponds to one of the 12 edges in Figure 28. These bits take a value of zero if their corresponding edge is not intersected by the isosurface and one otherwise. On the other hand, triTable also contains 256 entries consisting of up to 16 numbers that indicate how the required triangle facets are assigned to the triangle vertices.

In the previous example, only the third voxel is below the isosurface level. Thus, `cubeindex` takes the binary value 00001000. The first lookup table, edgeTable, indexed by the value of `cubeindex` (0000 1000 binary = 8 decimal) returns 1000 0000 1100. That binary value means edges 2, 3, and 11 are intersected by the isosurface, in agreement with Figure 28. The actual points where the edges are intersected are computed by linear interpolation of their corresponding voxel intensities, as shown in (Bourke, 1994). Now, the value of `cubeindex` is also used as an index to the second lookup table called triTable. Its eight element returns {3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, meaning that there is only one triangle facet assigned to edges 3, 11, and 2, again in agreement with Figure 28.

The procedure detailed above is executed by every CUDA thread, with one thread per voxel.

6.4 Integrating CUDA marching cubes with the ImmersiveTouch software

For the purpose of interactively simulating the process of bone drilling, the CUDA Marching Cubes algorithm in the previous section has been adapted and complemented with the volume haptics algorithm described in Chapter 3. For graphics rendering, the

CUDA Marching Cubes implementation is able to recompute the modified bone surface (to account for removed material) at interactive frame rates. On the other hand, the volume haptics algorithm is able to operate correctly when voxels are modified or eliminated in the original dataset. Therefore, the combination of Marching Cubes with the volume haptics algorithm, both taking as their input the modifiable voxel dataset, makes it possible to simulate bone removal operations at interactive frame rates.

6.4.1 Vertex Buffer Objects in Coin3D

Since version 2.5, Coin3D is able to use a standard mechanism in OpenGL known as Vertex Buffer Objects, or VBOs (Vertex array and VBO rendering in Coin). In essence, VBOs are used to store vertex coordinates and normal vectors in GPU memory, avoiding successive transfers from CPU memory space to GPU memory. As a result, significant performance gains are achieved in graphics rendering by eliminating those memory transfers.

The Marching Cubes implementation in CUDA expects to find two existing VBOs in memory to write its output. On one hand, the triangle vertex information generated by every thread is written to a 4-coordinate vertex VBO that may receive multiple elements per thread, as threads may generate vertex information for up to five triangle facets (see Figure 30). On the other hand, there must also be a VBO containing normal vectors, in which space is pre-allocated for one normal per vertex.

6.4.2 Rendering CUDA Marching Cubes data in Coin3D

Coin3D provides a class named SoIndexedFaceSet to handle generic indexed facesets (Coin3d SoIndexedFaceSet). Triangular faces are specified using the coordIndex

field. A very simple example of this class to show a single triangle facet would be (in OpenInventor .iv format):

```
#Inventor V2.1 ascii

     Coordinate3 {
        point [ 0 0 0, 1 0 0, 1 1 0 ]
     }

     IndexedFaceSet {
        coordIndex [ 0, 1, 2, -1 ]
     }
```

The Coordinate3 class contains three points in 3D space, defining a triangle. The coordIndex field in IndexedFaceSet specifies that vertices number 1, 2, and 3 are used to show the triangle. The fourth number (-1) is used as a special character to indicate the end of sequence.

In order for Coin3D to graphically render the polygonal data created by Marching Cubes, a derived class of its SoIndexedFaceSet class is used, along with SoCoordinate4 and SoNormal classes. The derived class is called SoCUDAIndexedFaceSet. As in the example above, the .iv representation of the scene for a single triangle would be:

```
#Inventor V2.1 ascii

  Coordinate4 {
      point [
         0 0 0 1, # 0
         1 0 0 1, # 1
         2 0 0 1, # 2
      ]
  }
  Normal {
      vector [
         0 0 1, # 0
         0 0 1, # 1
         0 0 1, # 2
      ]
  }
  CUDAIndexedFaceSet {
      coordIndex [
         0, 1, 2, -1,
      ]
  }
```

A similar construction to the above is used in our implementation. The number of points and normal vectors are initialized based on the maximum number of triangles expected for a given virtual model. Accordingly, one line per expected triangle is added in the coordIndex field.

As explained in the previous section, Coin3D will allocate VBOs to efficiently store the point and vector elements in the example above. To take advantage of Coin3D's VBOs, a mechanism has been implemented in the SoCUDAMarchingCubes class to catch at rendering time the internal pointers that Coin3D has allocated for its VBOs. In this way, these pointers to pre-allocated GPU memory space are passed to the CUDA Marching Cubes implementation, and therefore the triangle and normal vector information can be written directly to GPU memory by the CUDA kernel.

6.5 Burr-hole drilling simulation

A flow diagram of the algorithm for burr-hole drilling is shown in Figure 31. In the first blue box, the position of the haptic device in 3D space is determined and assigned to the variable CENTER. This variable contains the center of a bounding box delimiting the voxel to be evaluated by the algorithm.

TABLE VII

BURR-HOLE DRILLING ALGORITHM

ALGORITHM:	**Burr-hole drilling**	
INPUT :	`float RADIUS`	`radius of spherical region to drill`
	`vector devicePosition`	`current position of the haptic device`
OUTPUT :	`array voxels`	`modified voxels`

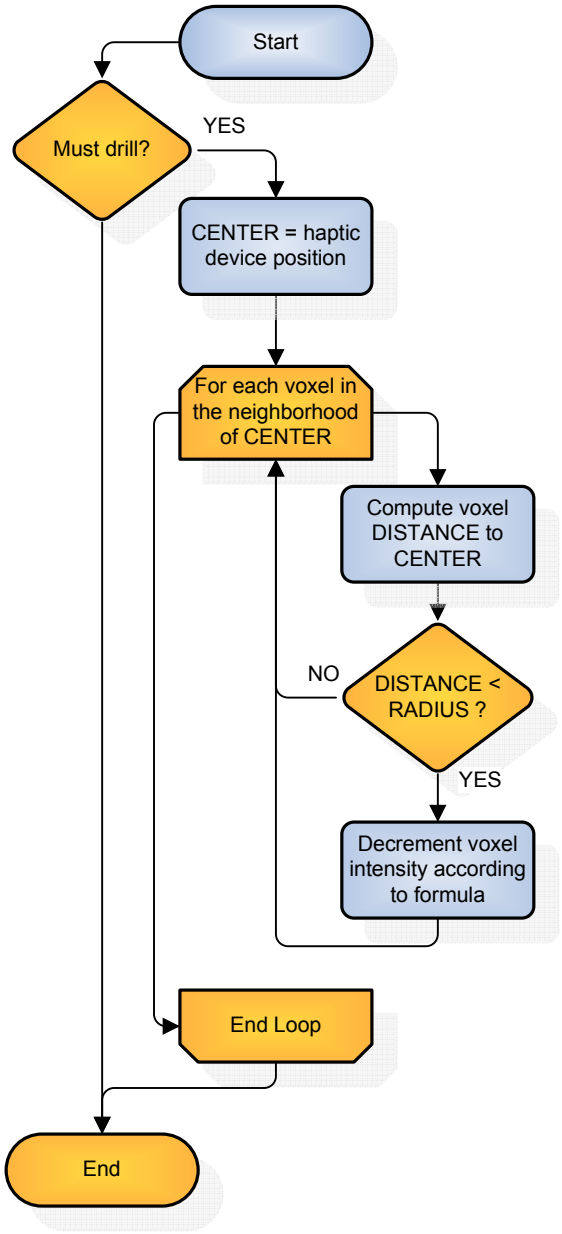


Figure 31. Burr-hole drilling algorithm.

For every candidate voxel, its 3D coordinates are obtained and its Euclidean distance to CENTER computed and stored in DISTANCE. With that, it is possible to determine whether the voxel lies inside a sphere with parameters given by CENTER and RADIUS. For all voxels inside that sphere, their intensities will be decremented according to the following expression:

$$newIntensity = oldIntensity - k \cdot \frac{RADIUS^2 - DISTANCE^2}{RADIUS^2}$$

where k is a constant used to adjust the cutting power of the drill. Figure 32 shows how the implementation of the drilling algorithm is visualized in the simulator.

Figure 32. Drilling a temporal bone model with a virtual matchstick burr.

6.6 Skin incision for ventriculostomy simulation

In order to expose the surface of the skull for drilling, it is necessary to create an incision in the virtual skin. If the virtual skin model is generated by the CUDA Marching Cubes algorithm, the incision can be simulated simply by discarding polygons around a given point. Therefore, with a simple modification to the original Marching Cubes kernel, every GPU thread generating triangles checks its distance to the CENTER of the desired incision, and in case their position lies inside a sphere of a given RADIUS, their generated polygons are discarded. The skin voxels in the same region are also eliminated to make the incision transparent to the haptic algorithm. The effect of this modification in the Marching Cubes CUDA kernel is shown in Figure 33.

Figure 33. Skin incision.

6.7 Craniotome cutting

A good approximation of the cutting effect of a virtual craniotome (Figure 21) can be modeled using an elliptical cylinder to describe the volume removed. An elliptical cylinder may be described by three parameters (Figure 34):

$$\begin{cases} a & \text{major axis of ellipse} \\ b & \text{minor axis of ellipse} \\ h & \text{height} \end{cases}$$

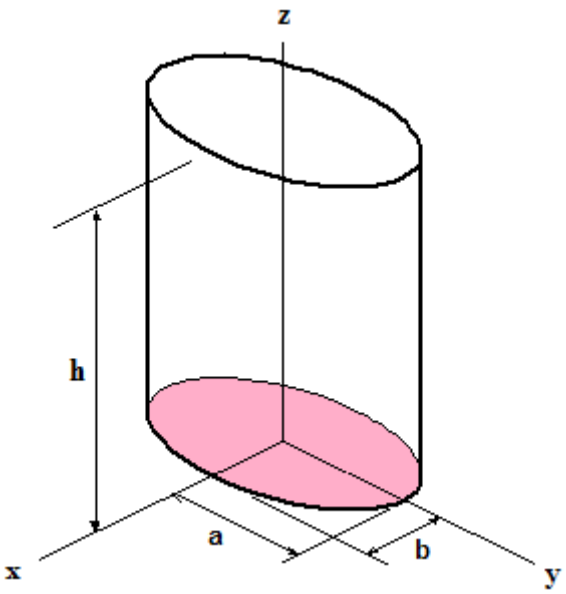


Figure 34. Parameters of an elliptical cylinder.

Given a point P in a plane parallel to the xy plane, it is important to determine whether P is contained in the ellipse defined by parameters a and b. From the definition of the ellipse we know that the distance between foci is:

$$f = \sqrt{a^2 - b^2}$$

The previous formula allows one to know the ellipse focal points F_1 and F_2 knowing only the major axis *a,* and minor axis *b*. Also, knowing the distance between point P and the focal points F_1 and F_2 , which we will denote PF_1 and PF_2 , a point P in the boundary or inside the ellipse will satisfy:

$$PF_1 + PF_2 \leq 2a$$

Therefore, specifying the ellipse only by its major axis *a,* and minor axis *b*, it is possible to test if an arbitrary point P is inside the ellipse. The same reasoning can be applied to voxels instead of points, where P now represents the voxel discrete coordinates in space.

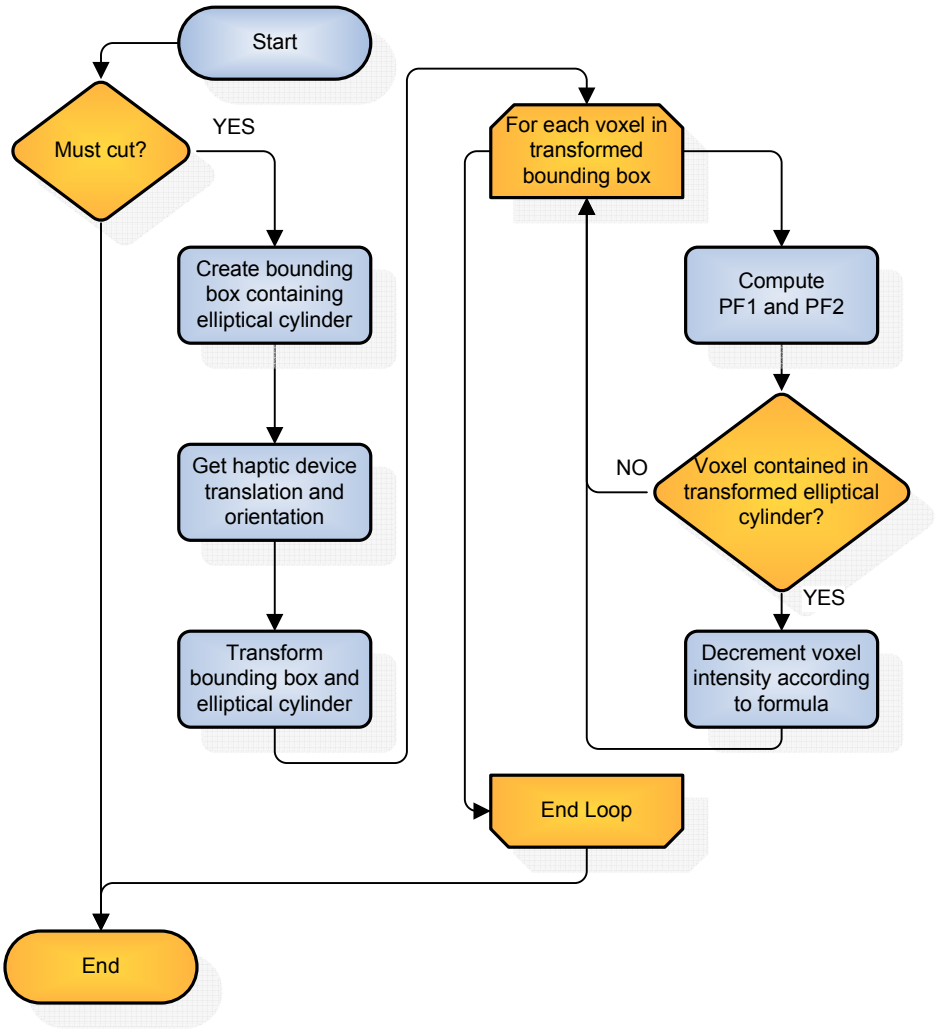


Figure 35. Craniotome cutting algorithm.

TABLE VIII

CRANIOTOME CUTTING ALGORITHM

ALGORITHM:	**Craniotome cutting**	
INPUT :	`float a,b,h`	`parameters of elliptical cylinder`
	`vector devicePosition`	`current position of the haptic device`
	`rotation deviceOrientation`	`current orientation of the haptic device`
OUTPUT :	`array voxels`	`modified voxels`

A flow diagram of the algorithm to model the cutting effect of a craniotome is presented in Figure 35. An elliptical cylinder is specified by parameters *a*, *b*, and *h* (Figure 34) to represent the region in space where voxel intensities will be modified to simulate volume removal. In order to position this elliptical cylinder in the appropriate region of 3D space, a bounding box containing the cylinder is created (first blue box in Figure 35). This bounding box is transformed according to the position and orientation of the haptic device to simulate the cut exactly where the virtual craniotome is located in 3D space (third blue box in Figure 35). With that, all voxels contained within the transformed bounding box can be evaluated to determine whether they lie inside the transformed elliptical cylinder. Taking into account the transformed parameters of the elliptical cylinder, those voxels that satisfy the condition $PF_1 + PF_2 \leq 2a$ will have their voxel intensities modified accordingly, achieving the material removal effect desired. An example of the effect created by the algorithm is appears in Figure 36 where the virtual craniotome is shown cutting a virtual model of the temporal bone region.

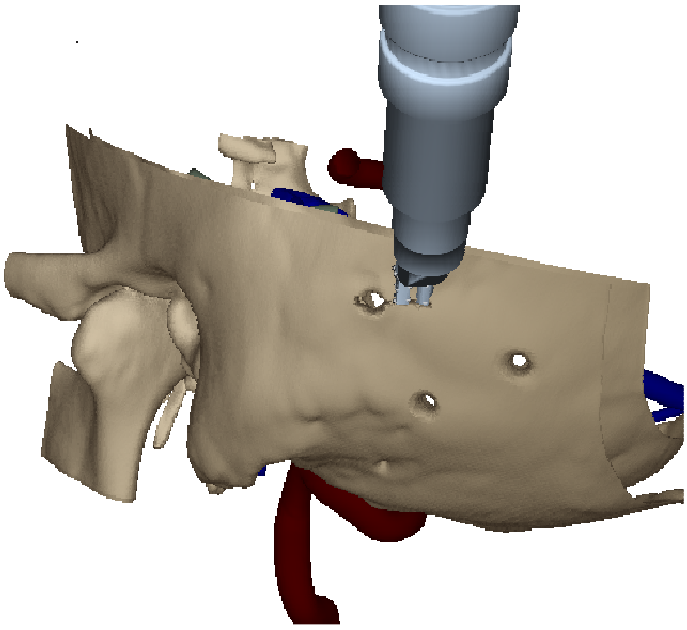


Figure 36. Modeling the cutting effect of a craniotome.

6.8 Contributions

One of the major contributions of the research presented in this chapter is that it makes possible to develop surgical simulation modules where graphical and haptic models are purely based on voxels. In addition, models with a modifiable voxel representation allow the simulation of bone-removal procedures at interactive frame rates, where the visuo-haptic representation of the removed voxels is recomputed on-the-fly.

7. SIMULATION MODULES

7.1 Ventriculostomy with burr-hole drilling

In the previous-generation ventriculostomy simulator (Luciano et al., 2006), burr-holes were pre-drilled in a location dictated by an experienced surgeon. The fact that it was not possible for the trainees to determine their own burr-hole location can be frustrating and, at the same time, might negatively affect their success rate in the procedure.

Using the techniques presented in Chapter 6, the original ventriculostomy simulation module has been extended in order to allow surgeons to determine the exact place in the surface of the skin where an incision should be made to drill a burr-hole in the skull.

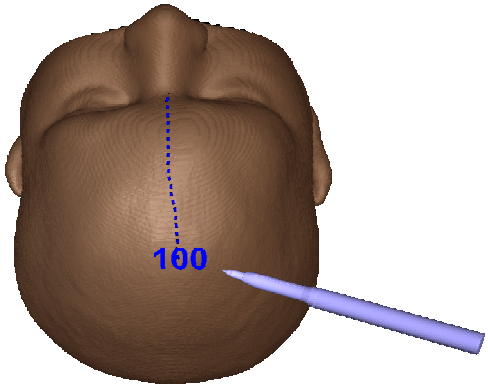


Figure 37. Using the marker tool (1).

In Figures 37 and 38, a virtual marker is used to determine the location of the skin incision, where the number in blue represents the length of the last trace in millimeters. In the example shown, the first trace is 100 mm (10 cm) along the midline (Figure 37), and

the second trace is 25 mm (2.5 cm) to the right. In order for the marker to create a visible trace, the surgeon needs to press a footpedal while touching the virtual skin with the marker.

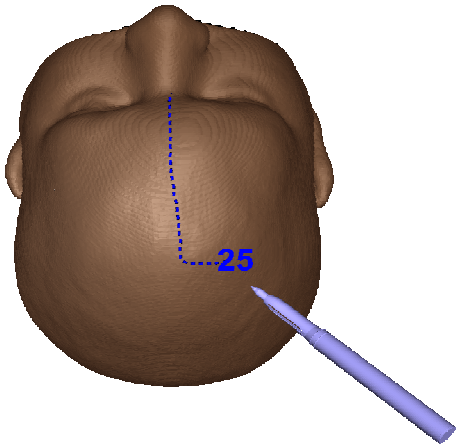


Figure 38. Using the marker tool (2).

The last point where the marker touched the skin is memorized by the computer. Once the surgeon is satisfied with the marked position, the blue trace can be erased and an incision can be created by pressing a key or using a remote control application on a mobile device (Figure 39).

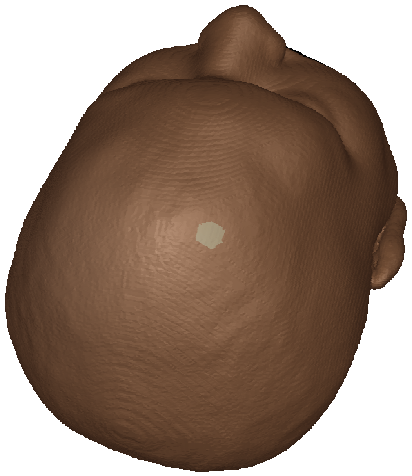


Figure 39. Creating an incision in the skin.

By pressing a second foot pedal, it is possible to switch the virtual instrument to a burr, which will be activated by pressing the first foot pedal. The surgeon is now able to progressively drill the burr-hole and feel the resistance of the bone, visualizing at the same time how the material is removed by the operation. A vibration haptic effect is activated when the drill foot pedal is pressed, increasing the realism of the drilling procedure.

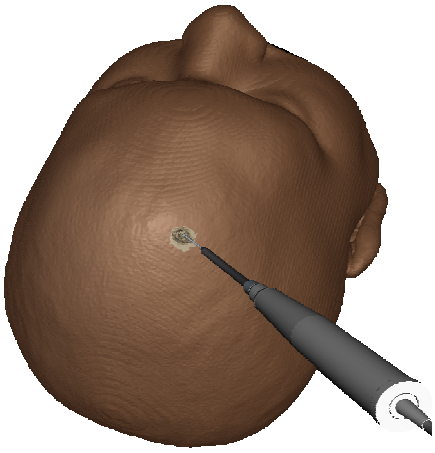


Figure 40. Drilling a burr-hole.

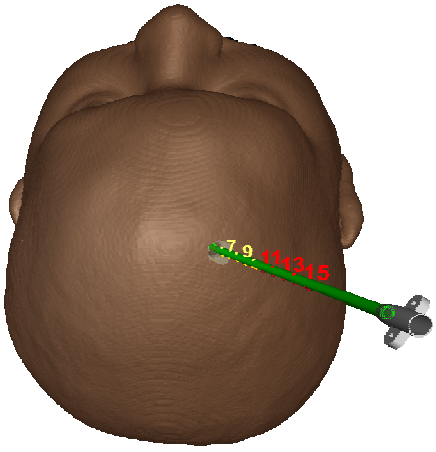


Figure 41. Successful ventricular cannulation.

Once the burr-hole is completed, the surface of the brain is visible and the next step is to perform the actual ventriculostomy. Figure 41 shows a simulation after the catheter has been successfully inserted into the virtual ventricles.

In some training scenarios, it is useful to allow the trainee to visualize the final location of the catheter inside the brain. This can be done using a virtual scissors tool that allows the interactive creation of a cut-away plane in the position and orientation desired by the surgeon. Figure 42 shows that the tip of the catheter is inside the ventricles, and therefore the procedure is considered successfully completed. A red sphere symbolizing the ideal target in the Foramen of Monro is also shown.

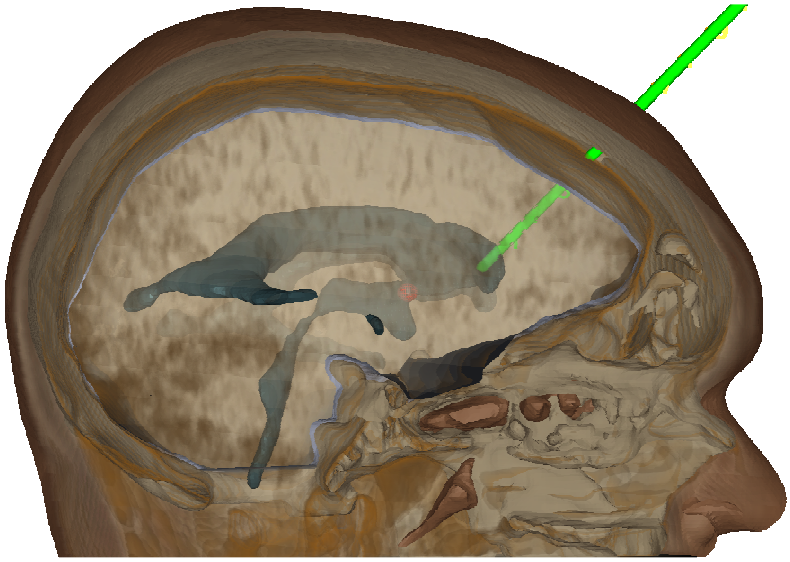


Figure 42. Cut-away plane showing final position of the catheter.

7.2 Percutaneous spine needle insertion with multipoint collision detection

The simulation module for percutaneous needle insertion described in (Luciano et al., 2013) was extended with the multipoint collision detection algorithm from Chapter 5. In the original implementation, the spine consisted of a polygonal mesh extracted from

CT scans using the techniques described in Chapter 3. With the extensions described in this thesis, the spine is now a volumetric model with CUDA Marching Cubes visualization, and the virtual Jamshidi needle has a multipoint representation that allows the surgeon to feel the contour of the pedicles and properly determine, by tactile feedback, the optimal entry point into the spine.

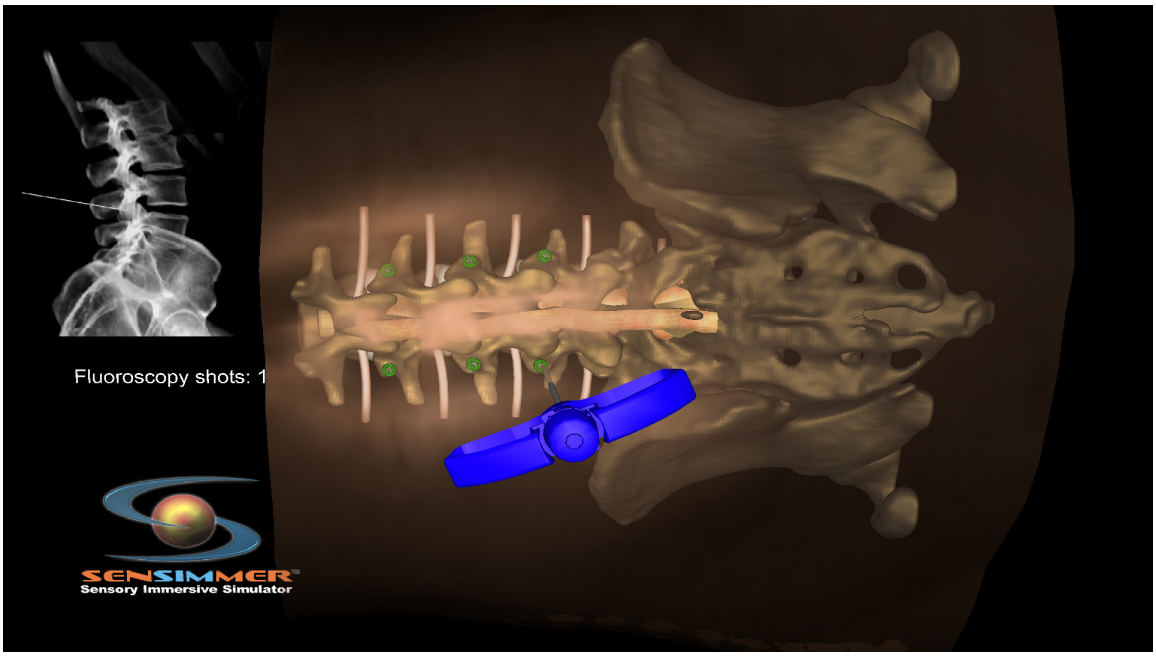


Figure 43. Percutaneous spine needle insertion.

Figure 43 presents a snapshot of the percutaneous needle simulation module. The virtual skin has been made transparent to show the virtual spine, though it is completely opaque during the simulation of the procedure. The green dots represent the ideal entry points into the pedicles. A virtual fluoroscopy simulation is also shown with which the surgeon can determine the optimal orientation and placement of the needle.

7.3 Subclavian central line placement with multipoint collision detection

Subclavian central line is another important procedure that benefits from the multipoint collision detection extensions. In this case, the needle must be advanced under and along the inferior border of the clavicle, for which it is essential that the virtual instrument supports multipoint collision detection.

A snapshot of the application is shown in Figure 44. Skin is a textured polygonal mesh, whereas the clavicle and other bone structures are voxel-based models generated by the CUDA Marching Cubes algorithm presented in Chapter 6. As in the percutaneous needle insertion module, the virtual needle is a multipoint-enabled instrument.

The software detects when the needle is inserted in the subclavian vein, switching the needle color to green to indicate success. On the other hand, if the trainee fails to reach the subclavian vein, the needle will turn red indicating failure.

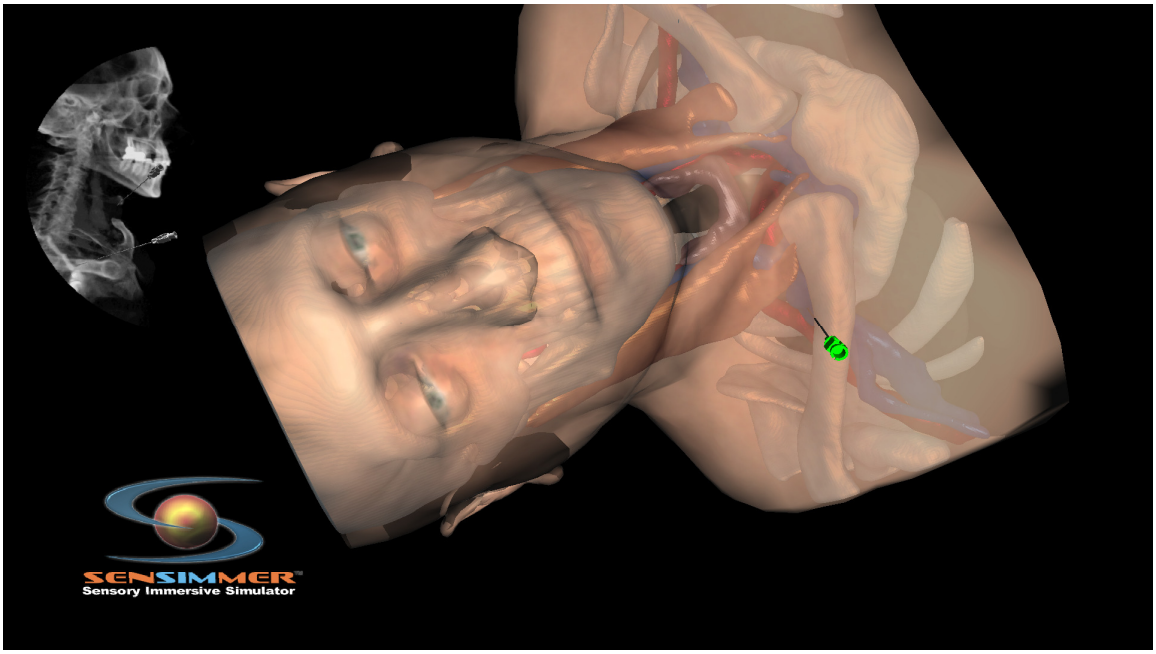


Figure 44. Subclavian central line simulation module.

8. VALIDATION EXPERIMENTS

In this chapter the results of validation experiments using some of the contributions in this thesis are presented, along with a summary of the journal publications where they have originally appeared.

8.1 Ventriculostomy experiments

8.1.1 Experiment 1

In this work, the impact of simulation-based practice of ventriculostomy with a library of virtual patients was studied. Neurosurgery resident's performance in simulated and real patients was evaluated.

TABLE IX

FIRST VENTRICULOSTOMY EXPERIMENT

Title:	*Practice on an Augmented Reality/Haptic Simulator and Library of Virtual Brains Improves Residents' Ability to Perform a Ventriculostomy*
Authors:	Rachel Yudkowsky, Cristian Luciano, Pat Banerjee, Alan Schwartz, Ali Alaraj, G Michael Lemole Jr, Fady Charbel, Kelly Smith, Silvio Rizzi, Richard Byrne, Bernard Bendok, David Frim
Journal:	Simulation in Healthcare 8 (1), Jan 2013, pp. 25-31

METHODS: CT scans of actual patients selected by Dr. Ali Alaraj at the Department of Neurosurgery, University of Illinois-Chicago Medical Center, were used. The techniques

presented in Chapter 2 were applied to create a library of 15 virtual patients for the ImmersiveTouch. Each patient model consisted of optimized polygonal meshes for skin, skull, brain, and ventricles. The virtual models represent a range of anatomies including normal, shifted, and compressed ventricles. Table X shows images of the patient library.

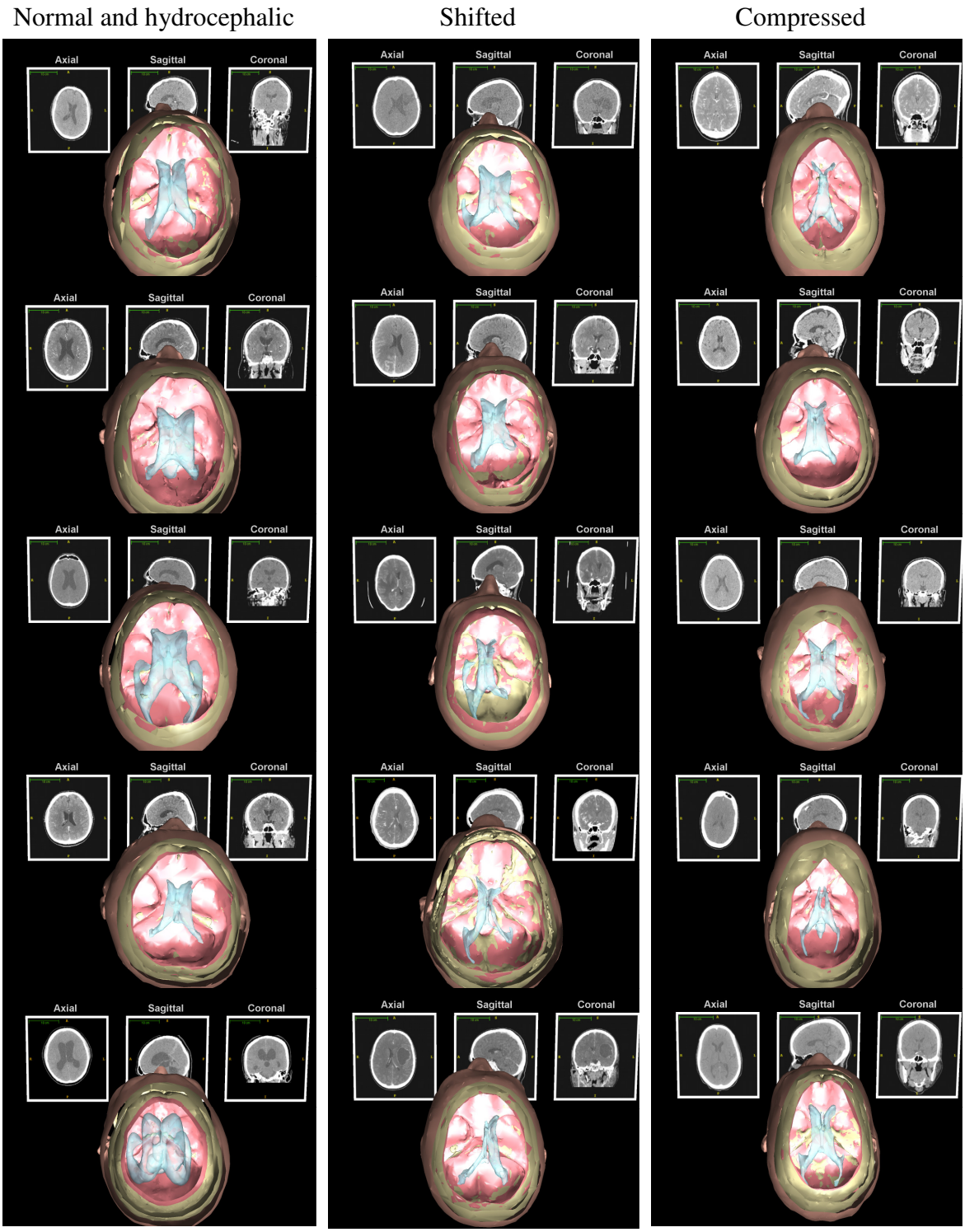
The ventriculostomy simulation application described in (Luciano et al., 2006) was used. Neurosurgery residents from the University of Illinois at Chicago, University of Chicago, Rush University Medical Center, and Northwestern University participated in individual simulator practice using the library of virtual patients. The protocol followed during the simulator experience has been described in detail in (Luciano, 2010). Performance of participants on novel brains in the simulator and during actual surgery (before and after intervention) was analyzed.

RESULTS: Simulator cannulation success rates increased after intervention, and live procedure outcomes showed improvement in the rate of successful cannulation on the first pass. However, the incidence of deeper, contralateral (simulator) and third-ventricle (live) placements increased after intervention. Residents reported that simulations were realistic and helpful in improving procedural skills such as aiming the probe, sensing the pressure change when entering the ventricle, and estimating how far the catheter should be advanced within the ventricle.

CONCLUSIONS: Simulator practice with a library of virtual brains representing a range of anatomies and difficulty levels may improve performance, potentially decreasing complications due to inexpert technique**.**

TABLE X.

PATIENT LIBRARY FOR VENTRICULOSTOMY



8.1.2 Experiment 2

The Congress of Neurological Surgeons (CNS) Simulation Committee developed a simulation-based curriculum incorporating the ImmersiveTouch simulator with the objective of enhancing resident training in ventriculostomy placement.

TABLE XI

SECOND VENTRICULOSTOMY EXPERIMENT

Title:	*Virtual Reality Based Simulation Training for Ventriculostomy: An Evidence Based Approach*
Authors:	Clemens M Schirmer, J Bradley Elder, Ben Roitberg, Darlene Angela Lobel
Journal:	Neurosurgery. Manuscript accepted for publication, May 2013.

METHODS: A course based neurosurgical simulation curriculum was introduced at the Neurosurgical Simulation Symposium at the 2011 and 2012 CNS annual meetings. A trauma module was developed to teach ventriculostomy placement as one of the neurosurgical procedures commonly performed in the management of traumatic brain injury. The course offered both didactic and simulator-based instruction, incorporating written and practical pre- and post-tests and questionnaires to assess improvement in skill level and validate the simulators as teaching tools. The ventriculostomy simulation module with burr-hole drilling, along with the voxel-based library of patients without pre-existing burr-holes (TABLE XII), was used in this work.

TABLE XII

VENTRICULOSTOMY LIBRARY USING VOXEL-BASED MODELS

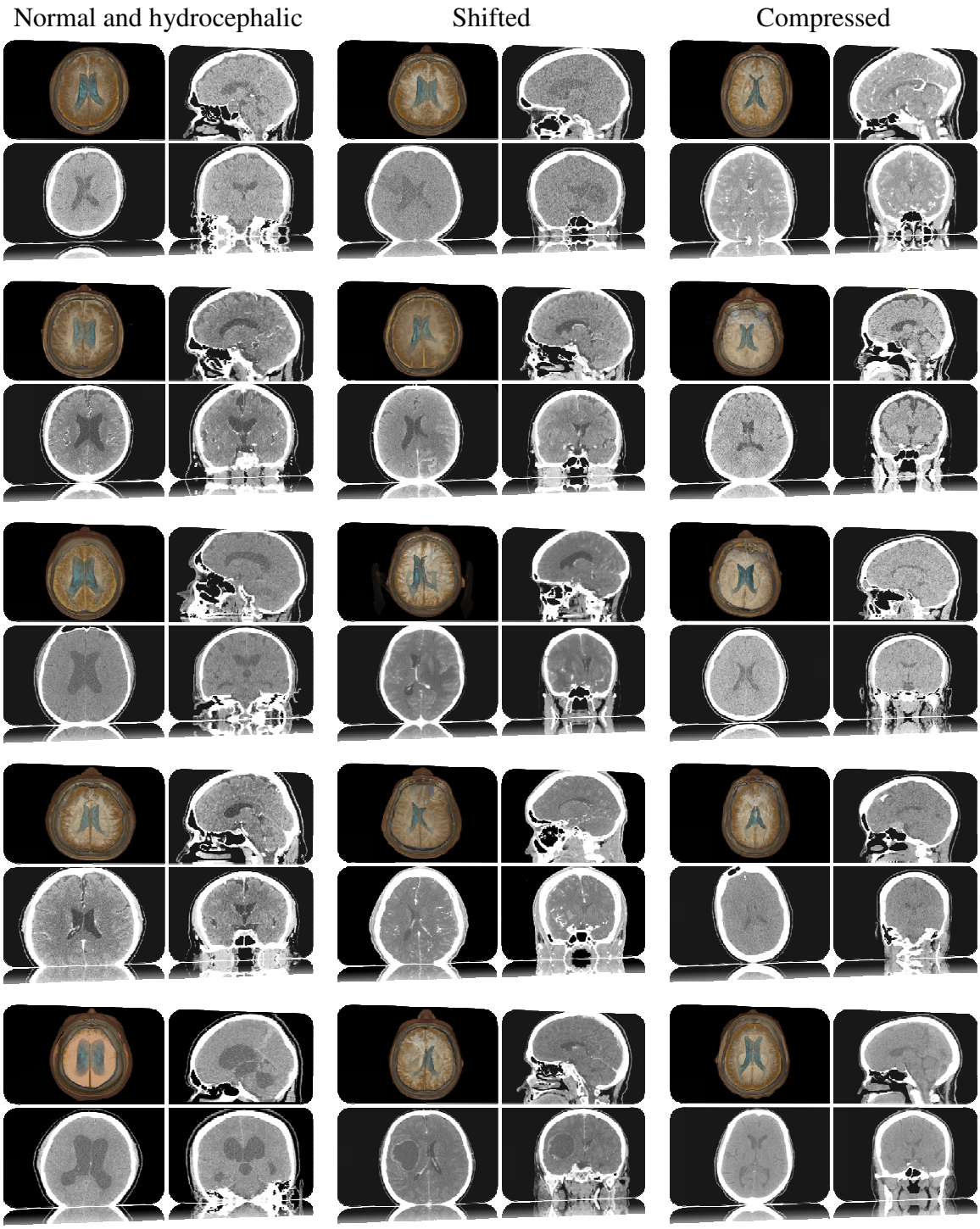


TABLE XIII

PERFORMANCE EVALUATION FOR SIMULATED VENTRICULOSTOMY

Score	1	2	3	4	5
Anatomy and landmarks	Did not define critical landmarks and structures		Identified either midline or coronal suture, but not both		Identified all critical landmarks and structures
Burr hole placement	Bur hole placed along midline or posterior to coronal suture		Bur hole placed too laterally or too anteriorly		Bur hole at Kocher's point
Trajectory of catheter insertion	Traverses midline or passes through critical structures		Too lateral or too medial, however avoids critical structures		Ideal path, avoiding all critical structures
Depth of catheter insertion	<3cm or >9cm		4-5cm or 6-7cm		5-6cm
Final location of catheter tip	Catheter not positioned in ventricle		Catheter in center of ipsilateral ventricle or in contralateral ventricle		Tip within 5mm of ipsilateral foramen of Monroe
Time to complete procedure	Unable to complete procedure in allotted time		10 minutes		Less than 5 minutes

Ordinal scores between 1 and 5 were given for each performance measure, scores 2 and 4 were interpolated between the endpoints given in the table for score 1, 3 and 5 (Courtesy Schirmer et al., 2013).

RESULTS: Seven participants completed the ventriculostomy simulation. Significant improvements were observed in anatomy ($p<0.04$), burr hole placement ($p<0.03$), final location of the catheter ($p=0.05$), and procedure completion time ($p<0.004$). Senior residents planned a significantly better trajectory ($p<0.01$) and junior participants

improved most in terms of identifying the relevant anatomy ($p<0.03$) and the time required to complete the procedure ($p<0.04$).

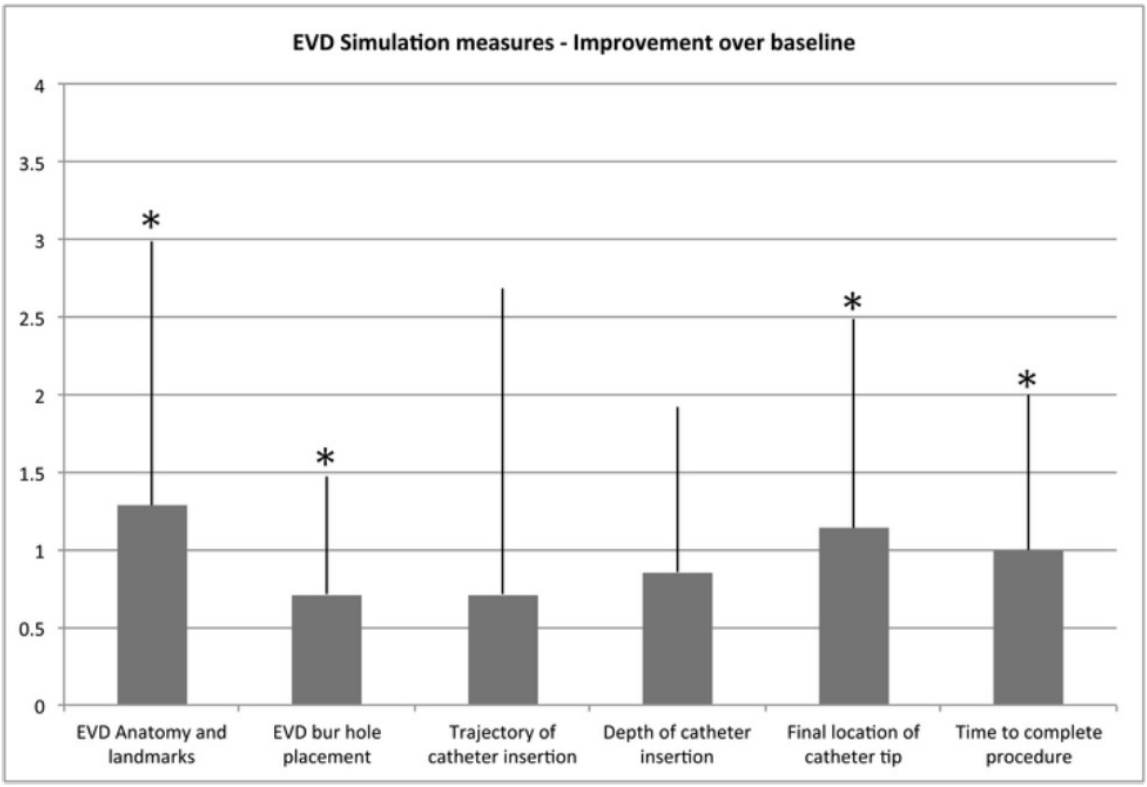


Figure 45. Improvement over baseline in ventriculostomy (Schirmer et al., 2013).

CONCLUSIONS: Virtual ventriculostomy placement as part of the CNS simulation trauma module complements standard training techniques for residents in the management of neurosurgical trauma. Improvement in didactic and hands-on knowledge by course participants demonstrates the usefulness of the ImmersiveTouch as a training tool.

8.2 Pedicle screw experiments

8.2.1 Experiment 1

The use of the ImmersiveTouch as a training tool for percutaneous spinal needle placement was considered in this study. The objective was to evaluate the learning effectiveness in terms of entry point/target point accuracy of percutaneous spinal needle placement using a simulation module that allows the user to control the duration of computer-simulated fluoroscopic exposure, thereby simulating the actual OR experience.

TABLE XIV

FIRST PEDICLE SCREW EXPERIMENT

Title:	*Percutaneous spinal fixation simulation with virtual reality and haptics*
Authors:	Cristian J Luciano, P Pat Banerjee, Jeffery M Sorenson, Kevin T Foley, Sameer A Ansari, Silvio Rizzi, Anand V Germanwala, Leonard Kranzler, Prashant Chittiboina, Ben Z Roitberg
Journal:	Neurosurgery 72 (Supplement 1), January 2013, pp A89-A96

METHODS: Sixty-three fellows and residents performed needle placement on the simulator during the 2010 American Association of Neurosurgical Surgeons (AANS) annual meeting. A virtual needle was percutaneously inserted into a virtual patient's thoracic spine derived from an actual patient CT data set.

RESULTS: Ten of 126 needle placement attempts by 63 participants ended in failure for a failure rate of 7.93%. From all 126 needle insertions, the average error went from 15.69

in the first attempt to 13.91 in the second attempt. The following figure shows the distribution of performance error in the first and second attempts:

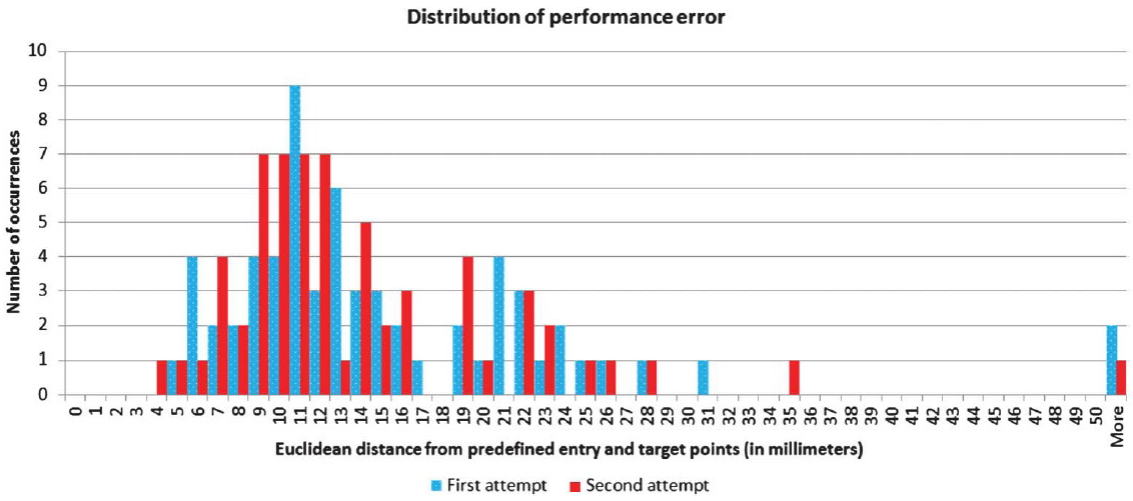


Figure 46. Distribution of performance error.

Similarly, the average fluoroscopy exposure improved from 4.6 in the first attempt to 3.92 in the second attempt. The next figure shows the distribution:

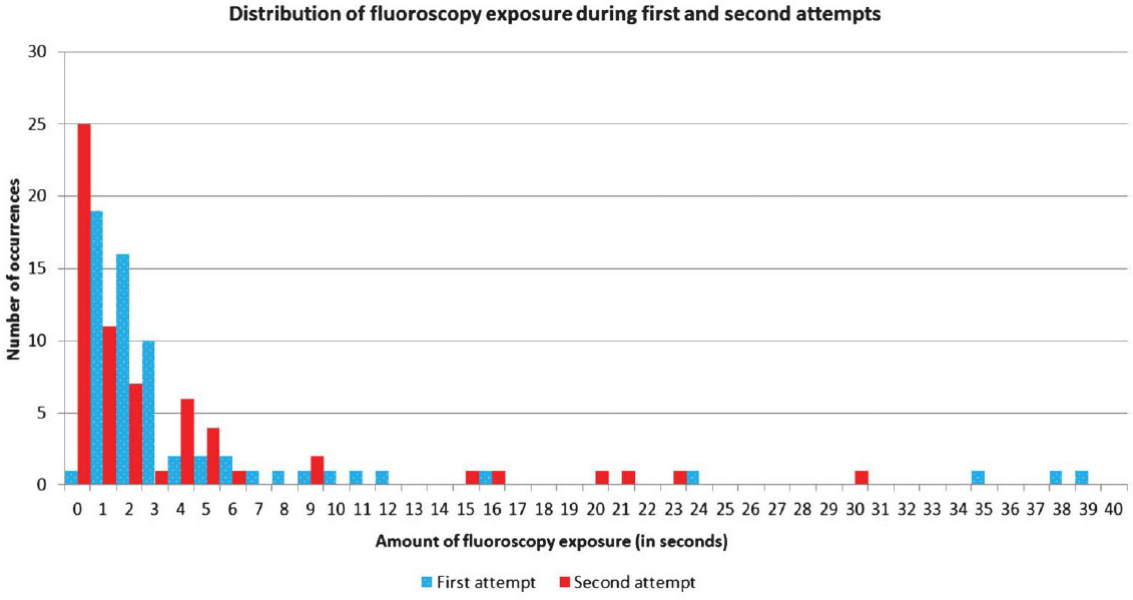


Figure 47. Distribution of fluoroscopy exposure.

Finally, the average individual performance score also improved from the first to the second attempt (32.39 vs. 30.71)

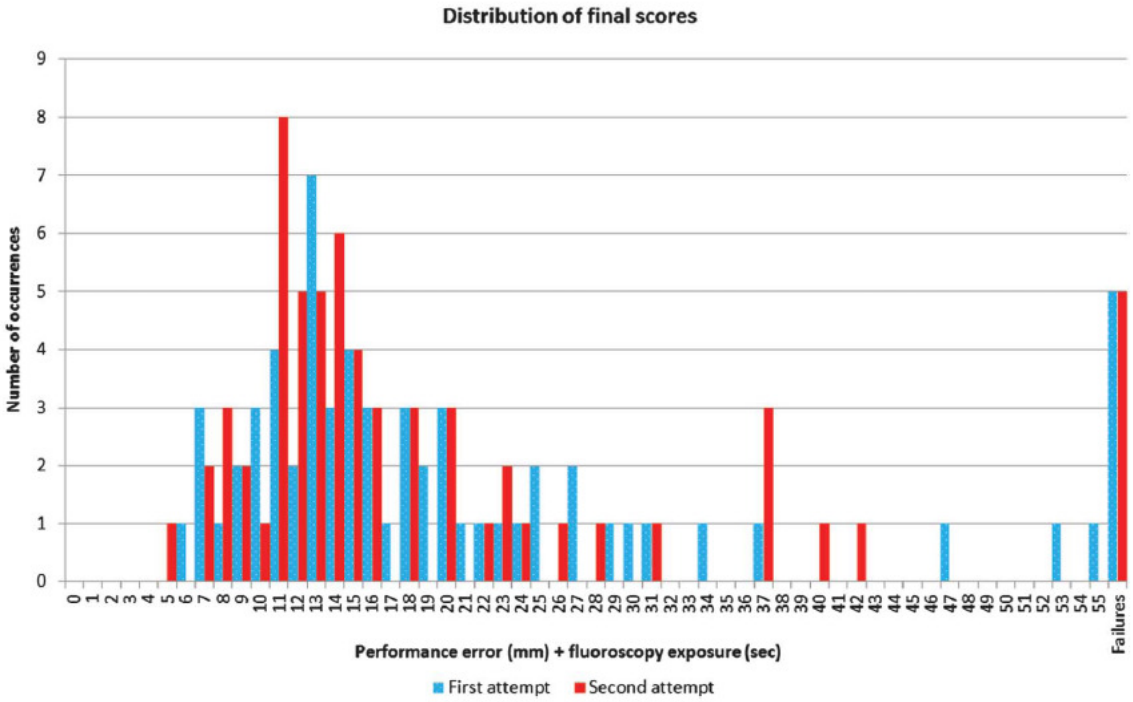


Figure 48. Distribution of final scores.

Performance accuracy yielded P =.04 from a 2-sample t test in which the rejected null hypothesis assumes no improvement in performance accuracy from the first to second attempt in the test session.

CONCLUSION: The experiments showed evidence (P =.04) of performance accuracy improvement from the first to the second percutaneous needle placement attempt. This result, combined with previous learning retention and/or face validity results of using the simulator for open thoracic pedicle screw placement and ventriculostomy catheter placement, supports the efficacy of augmented reality and haptics simulation as a learning tool.

8.2.2 Experiment 2

The ability of a pedicle screw simulation module on the ImmersiveTouch to improve screw placement accuracy in sawbone models was explored in this work. Participants were senior medical students with a single training session.

TABLE XV

SECOND PEDICLE SCREW EXPERIMENT

Title:	*Computer-simulation training positively impacts the accuracy of pedicle screw placement performed by aspiring neurosurgery residents in sawbone models*
Authors:	Jaime Gasco, Achal Patel, Juan Ortega-Barnett, Daniel Branch, Yong Fan Kuo, Cristian Luciano, Silvio Rizzi, Patrick Kania, Martin Matulyauskas, Pat Banerjee, Ben Z. Roitberg
Journal:	World Neurosurgery, Manuscript accepted for publication. May 2013

METHODS: Thirty-eight applicants to neurosurgery residency were offered anonymous participation in the study, and randomized into 3 groups prior to the placement of two lumbar pedicle screws in a sawbone model. The groups were: (A) Control – no prior simulation; (B) Simulation of pedicle finder insertion in a 3-D vertebra; and (C) Lumbar pedicle screw insertion within a surgical environment. The sawbone models then underwent CT imaging and triplanar analysis to detect errors in screw coronal entry point, axial and sagittal deviations, length error, and pedicle breach. The screw placement was further classified into acceptable ($\leq$ 2 errors) or not acceptable ($\geq$ 3 errors) based on the above variables. The overall performance in each group was based on the mean

number of errors per screw. The Kruskal-Wallis test was used to determine any significance of difference using an adjusted threshold p-value of 0.0169 (Bonferroni method).

RESULTS: A total of 76 pedicle screws were analyzed. Group B (pedicle finder simulation), improved performance by 24.0% (p=0.1505) vs. group A (no simulation); Group C (open pedicle screw simulation) improved by 53.8% (p = 0.0005) vs. group A and 39.2% (p = 0.0078) vs. group B. Reductions in the number of unacceptable screws was 17.9% and 26.9% for groups B and C respectively relative to group A, as shown in the following table.

TABLE XVI

ACCEPTABLE VS NON-ACCEPTABLE PEDICLE SCREWS

	ACCEPTABLE (N,%)	NOT ACCEPTABLE (N,%)	RELATIVE CHANGE ACCEPTABLE (%)	MEAN NO. OF ERRORS /SCREW	STANDARD DEVIATION	N
GROUP A	17 (65.4%)	9 (34.6%)	0	2.08	1.23	26
GROUP B	20 (83.3%)	4 (16.7%)	17.9	1.58	0.88	24
GROUP C	24 (92.3%)	2 (7.7%)	26.9	0.96	0.96	26

CONCLUSIONS: Computer-simulation training positively impacts the accuracy of pedicle screw placement performed by neurosurgery applicants in sawbone models with only a single simulated practice compared to individuals with no prior simulation exposure.

9. FINAL CONCLUSIONS

This thesis presents a careful balance of research, development, and rigorous scientific methodology. Within the scope of surgical simulation, multiple contributions have been presented, namely:

- A method to generate patient-specific polygonal mesh 3D models for haptics and graphics representation, containing an optimal number of polygons, and obtained by a well-specified sequence of operations applied to the original data. Results were published in the 2007 IEEE International Conference on Automation, Science and Engineering (Rizzi et al., 2007).
- An algorithm for voxel-based 3-DOF haptic feedback that extends the OpenHaptics library, and overcomes the disadvantages of other similar algorithms. Results were published in the 2010 IEEE Haptics Symposium (Rizzi et al., 2010).
- A number of scientific experiments evaluating performance of haptics-graphics combinations of algorithms, where it was proved that polygonal mesh graphics rendering, along with the voxel-based haptics algorithm, exhibit the best performance and stability in terms of combined rendering time. Results were published in the ASME Journal of computer and Information Science in Engineering (Rizzi et al., 2012).
- Extensions to the original volume haptics algorithm for object-to-object collision detection.

- A fast implementation of the Marching Cubes in GPU integrated with the *ImmersiveTouch*® application framework for interactively recomputing a polygonal mesh from its voxel representation.
- Multiple algorithms for bone-removal procedures, including burr-hole drilling and craniotome cut.
- Implementation of surgical simulation modules using the previously described algorithms. Modules include ventriculostomy with burr-hole drilling, percutaneous spine needle insertion, and subclavian central line.
- Participation in validation experiments involving the outcome of the research and development presented in this thesis. Promising and encouraging results were obtained as a result of these experiments.

CITED LITERATURE

Adachi, Y., Kumano, T., Ogino, K.: Intermediate representation for stiff virtual objects. Proceedings of Virtual Reality Annual International Symposium, 1995, pp.203-210.

Avila, R.S., Sobierajski, L.M.: A haptic interaction method for volume visualization. In Proceedings of the 7th Conference on Visualization '96 (San Francisco, California, United States, October 28 - 29, 1996). R. Yagel and G. M. Nielson, Eds. IEEE Visualization. IEEE Computer Society Press, Los Alamitos, CA, 197-ff.

Banerjee, P., Charbel, F.: On-Demand High Fidelity Neurosurgical Procedure Simulator Prototype at University of Illinois using Virtual Reality and Haptics. Accreditation Council for Graduate Medical Education (ACGME) Bulletin, September 2006; p. 20-21.

Banerjee, P., Luciano, C., Florea, L., Dawe, G., Steinberg, A., Drummond, J., Zefran, M.: Compact Haptic and Augmented Virtual Reality System. Board of Trustees University of Illinois, U.S. Patent 11/338434, 2010.

Basdogan, C., Ho, C., Srinivasan, M.A.: A Ray-Based Haptic Rendering Technique for Displaying Shape and Texture of 3D Objects in Virtual Environments. The Winter Annual Meeting of ASME'97, DSC-Vol. 61, pp. 77-84, Dallas, TX, Nov. 16-21, 1997.

Basdogan, C., Laycock, S.D., Day, A.M., Patoglu, V., Gillespie, R.B.: 3-Dof Haptic Rendering. In Haptic Rendering, Eds: M.C. Lin and M. Otaduy, Publisher: A.K. Peters, pp. 311-333, 2007.

Bourke, P.: Polygonising a Scalar Field, May 1994. Available: http://paulbourke.net/geometry/polygonise/

Caselles,V., Kimmel, R., Sapiro, G.: Geodesic Active Contours. Int. J. Comput. Vis. 22, 61 - 79. 1997.

Cebral, J., Löhner, R.: From medical images to anatomically accurate finite element grids. Int. J. Numer. Meth. in Engng, 2001; 51:985-1008.

CHAI3D, available: http://www.chai3d.org

Chen, K., Heng, P., Sun, H.: Direct haptic rendering of isosurface by intermediate representation. In Proceedings of the ACM Symposium on Virtual Reality Software and Technology (Seoul, Korea, October 22 - 25, 2000). VRST '00. ACM, New York, NY, 188-194.

Du, J., Yang, X., Du, Y.: From Medical Images to Finite Grids System. Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, Sept. 2005.

Geiss, R.:Generating Complex Procedural Terrains Using the GPU, GPU Gems 3, NVIDIA Corporation, Addison-Wesley, 2007. Available: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html

Gibson, S.F.: Beyond Volume Rendering: Visualization, Haptic Exploration, and Physical Modeling of Voxel-based Objects, Mitsubishi Electric Research Laboratories, Technical Report 95-04, 1995.

H3D, available: http://www.h3dapi.org/

Itkowitz, B., Handley, J., Zhu, W.: The OpenHaptics toolkit: a library for adding 3D Touch navigation and haptics to graphics applications. Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005, pp. 590-591, 18-20 March 2005.

ITK-SNAP. Available: http://www.itksnap.org/

Ito, Y., Shum, P., Shih, A., Soni, B., Nakahashi, K.: Robust generation of high-quality unstructured meshes on realistic biomedical geometry. Int. J. Numer. Meth. in Engng, 2006; 65:943-973.

Iwata, H., Noma, H.: Volume haptization. Virtual Reality, 1993. Proceedings IEEE 1993 Symposium on Research Frontiers, pp.16-23, 25-26 Oct 1993.

Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. International Journal of Computer Vision, 1(4), 1987.

Körner, O., Schill, M., Wagner, C., Bender, H.J., Männer, R.: Haptic volume rendering with an intermediate local representation. In Proceedings of the 1st International Workshop on Haptic Devices in Medical Applications, 1999, pp. 79–84.

Lorensen, W., Cline, H.: Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics, Vol. 21, No. 4, July 1987.

Luciano, C., Banerjee, P., Florea, L., Dawe, G.: Design of the ImmersiveTouch™: A High-Performance Haptic Augmented VR System. Proceedings of Human-Computer Interaction (HCI) International Conf., Las Vegas, 2005.

Luciano, C., Banerjee, P., Lemole, G.M., Charbel, F.: Second Generation Haptic Ventriculostomy Simulator Using the ImmersiveTouch™ System. Proceedings of 14th Medicine Meets Virtual Reality, J.D. Westwood et al. (Eds.), IOSPress, pp. 343-348, 2006.

Luciano, C. J.: Open Surgery Training Simulator Using Haptics and Augmented Reality Technologies. Submitted as partial fulfillment of the requirements for the degree of Doctor of Philosophy in Industrial Engineering and Operations Research, Graduate College, University of Illinois at Chicago, August 2010.

Lundin, K., Cooper, M., Persson, A., Evestedt, D., Ynnerman, A.: Enabling design and interactive selection of haptic modes. Virtual Reality, 2006.

Lundin, K., Cooper, M., Ynnerman, A.: The orthogonal constraints problem with the constraint approach to proxy-based volume haptics and a solution. In Proceedings of SIGRAD Conference, pp. 45-49, Lund, Sweden, Nov. 2005.

Lundin, K., Cooper, M., Ynnerman, A.:Haptic Rendering of Dynamic Volumetric Data. IEEE Transactions on Visualization and Computer Graphics, vol.14, no.2, pp.263-276, March-April 2008.

Lundin, K., Gudmundsson, B., Ynnerman, A.: General proxy-based haptics for volume visualization. Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005, pp. 557-560, 18-20 March 2005.

Lundin, K., Ynnerman, A., Gudmundsson, B.: Proxy-based haptic feedback from volumetric density data. In Proceedings of the Euro-haptic Conference, pp. 104-109. University of Edinburgh, United Kingdom, 2002.

Lundin, K. : Direct Volume Haptics for Visualization. PhD thesis, Linköping University, 2007a.

Lundin, K.: Fast and High Precision Volume Haptics. EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007, pp.501-506, 22-24 March 2007b.

Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M., Taylor, R.M.: Adding force feedback to graphics systems: issues and solutions. In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96. ACM, New York, NY, 447-452. 1996.

Massie, T.H., Salisbury, J.K.: The PHANTOM Haptic Interface: A Device for Probing Virtual Objects. Symp. On Haptic Interfaces for Virtual Environments. Chicago, IL, Nov. 1994.

McNeely, W.A., Puterbaugh, K.D., Troy, J.J.: Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling, SIGGRAPH '99 Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999, pp 401-408.

Melonakos, J., Al-Hakim, R., Fallon, J., Tannenbaum, A.: Knowledge-Based Segmentation of Brain MRI Scans Using the Insight Toolkit. The Insight Journal - ISC/NA-MIC/MICCAI Workshop on Open-Source Software, 2005.

Morris, D., Sewell, C., Barbagli, F., Salisbury, K., Blevins, N.H., Girod, S.: Visuohaptic Simulation of Bone Surgery for Training and Evaluation. Computer Graphics and Applications, IEEE , vol.26, no.6, pp. 48-57, Nov.-Dec. 2006.

OpenHaptics Toolkit Version 3.0 Programmer's Guide [a], pp.7-6.

OpenHaptics Toolkit Version 3.0 Programmer's Guide [b], pp. 7-33.

Petersik, A., Pflesser, B., Tiede, U., Hohn., K.H., Leuwer, R.: Haptic Volume Interaction with Anatomic Models at Sub-Voxel Resolution, Proc IEEE VR, Orlando, FL, Mar 2002.

Prabhu, S., Zauner, A., Bullock, M.: Surgical Management of Traumatic Brain Injury. Youmans Neurological Surgery, 5th. edition, WB Saunders, Philadelphia, 2004, pp. 5145-5180.

Rizzi, S., Luciano, C., Banerjee, P.: Haptic Interaction with Volumetric Datasets Using Surface-Based Haptic Libraries. Haptics Symposium, 2010 IEEE , vol., no., pp.243-250, 25-26 March 2010.

Rizzi, S., Banerjee, P., Luciano, C.: Automating the Extraction of 3D Models from Medical Images for Virtual Reality and Haptic Simulations. Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on , pp.152-157, Sept. 2007.

Ruffaldi, E., Morris, D., Edmunds, T., Barbagli, F., Pai, D.: Standardized Evaluation of Haptic Rendering Systems. 14th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2006, pp. 225- 232, 25-26 March 2006.

Ruspini, D.C., Kolarov, K. , Khatib, O.: The haptic display of complex graphical environments. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 345-352, 1997.

Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics. Kitware, Inc., 2006.

SensAble Haptic Devices, available: http://www.sensable.com/products-haptic-devices.htm

SensAble Technologies OpenHaptics, available at http://www.sensable.com/products-openhaptics-toolkit.htm

Schirmer, C.M., Elder, J.B., Roitberg, B., Lobel, D.A.: Virtual Reality Based Simulation Training for Ventriculostomy: An Evidence Based Approach. Neurosurgery, manuscript accepted for publication, May 2013.

Srimathveeravalli, G., Gourishankar, V., Kumar, A., Kesavadas, T.: Experimental Evaluation of Shared Control for Rehabilitation of Fine Motor Skills. J. Comput. Inf. Sci. Eng. 9, 014503, 2009.

Systems In Motion Coin3D, available: http://www.coin3d.org/

The Insight Segmentation and Registration Toolkit. Available: http://www.itk.org/

The Visible Human Project. Available: http://www.nlm.nih.gov/research/visible/visible_human.html

The Visualization Toolkit. Available: http://www.kitware.com

Vertex array and VBO rendering in Coin. Available: http://doc.coin3d.org/Coin/vbo__rendering.html

Wolf, I., Vetter, M., Wegner, I., Nolden, M., Böttger, T., Hastenteufel, M., Schöbinger, M., Kunert, T., Meinzer, H.: The Medical Imaging Interaction Toolkit (MITK) - a toolkit facilitating the creation of interactive software by extending VTK and ITK. Proc. SPIE Int. Soc. Opt. Eng. 5367, 16, 2004.

Young, P., Tabor, G., Collins, T., Richterova, J., Dejuniat, E., Beresford-West, T.: Automating the generation of 3D finite element models based on medical imaging data. Digital Human Modeling for Design and Engineering Conference, Lyon, July 2006.

Yudkowsky, R., Luciano, C., Banerjee, P., Alaraj, A., Lemole, M., Schwartz, A., Charbel F., Mlinarevich, N., Smith, K., Gandhi, S., Rizzi, S.: A library of virtual brains for ventriculostomy practice on a VR/haptic simulator - initial validity evidence. Simulation in Healthcare 2010; 5:43

Yudkowsky, R., Luciano, C., Banerjee, P., Alaraj, A., Lemole, M., Schwartz, A., Charbel F., Smith, K., Rizzi, S.: Ventriculostomy Practice on a Library of Virtual Brains using a VR/Haptic Simulator Improves Simulator and Surgical Outcomes. Poster presented at the 12th Annual International Meeting on Simulation in Healthcare, San Diego CA, January 2012.

Yushkevich, P., Piven, J., Cody, H., Ho, S., Gerig, G.: Geodesic Snakes for User-Guided Segmentation of 3-D Anatomical Objects: Significantly Improved Efficiency and

Reliability. Jan. 2005. Available: http://www.itksnap.org/~paul/files/docs/yushkevich05snap.pdf

Yushkevich, P., Piven, J., Hazlett, H., Smith, R., Ho, S., Gee, J., Gerig, G.: User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. Neuroimage 31, 2006, pp.1116-1128.

Zhu, S., Yuille, A.: Region competition: unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol.18, Iss.9, Sep 1996, 884-900.

Zilles, C.B., Salisbury, J.K.: A constraint-based god-object method for haptic display. Intelligent Robots and Systems 95. IEEE/RSJ International Conference on 'Human Robot Interaction and Cooperative Robots'. 1995, pp.146-151 vol.3, 5-9 Aug 1995.

VITA

NAME: Silvio Rizzi

EDUCATION:

Ph.D., Industrial Engineering and Operations Research
University of Illinois at Chicago (UIC), Illinois, 2013

Master of Science, Electrical and Computer Engineering
University of Illinois at Chicago (UIC), Illinois, 2006

Bachelor of Science, Electronics Engineering
Universidad Tecnológica Nacional (UTN), Mendoza, Argentina, 2002

RESEARCH EXPERIENCE:

Summer 2006 – Summer 2013:
Research Assistant
Department of Mechanical and Industrial Engineering, UIC, IL

TEACHING EXPERIENCE:

Fall 2012:
Lecturer – IE464 Virtual Automation
Department of Mechanical and Industrial Engineering, UIC, IL

Fall 2007, Spring 2008, Fall 2009, Spring 2010:
Teaching Assistant – IE201 Financial Engineering
Department of Mechanical and Industrial Engineering, UIC, IL

HONORS:

First Place Research Abstract Award: R. Yudkowsky, C. Luciano, P. Banerjee, A. Alaraj, M. Lemole, A. Schwarz, F. Charbel, K. Smith, **S. Rizzi**, "Ventriculostomy Practice on a Library of Virtual Brains Using a VR/Haptic Simulator Improves Simulator and Surgical Outcomes," 12th Annual International Meeting on Simulation in Healthcare, San Diego, CA, Jan 2012

Fulbright Scholarship (2004-2006)

CONFERENCE PUBLICATIONS:

Cristian Federico Perez Monte, Fabiana Piccoli, Cristian Luciano, **Silvio Rizzi**, German Bianchini, Paola Caymes Scutari, "*Estimation of Volume Rendering Efficiency with GPU in a Parallel Distributed Environment*," 6th Workshop on "Biomedical and Bioinformatics Challenges for Computer Science" (BBC 2013), International Conference on Computational Science, Barcelona, Spain, June 2013.

Silvio Rizzi, Cristian Luciano, Pat Banerjee, "*Haptic interaction with volumetric datasets using surface-based haptic libraries*," IEEE Haptics Symposium, 2010 IEEE, 243-250, 2010.

P Pat Banerjee, Shaojie Zhang, Cristian Luciano, **Silvio Rizzi**, "*Remote Exercise and Game Architecture Language (REGAL)*," Proc. Rectech 2nd State of the Science Conference, 53, 2010.

Silvio Rizzi, Pat Banerjee, Cristian Luciano, "*Automating the extraction of 3d models from medical images for virtual reality and haptic simulations*," CASE 2007, IEEE International Conference on Automation Science and Engineering, 152-157, 2007.

Cristian Luciano, Pat Banerjee, **Silvio Rizzi**, "*GPU-based elastic-object deformation for enhancement of existing haptic applications*," CASE 2007, IEEE International Conference on Automation Science and Engineering, 146-151, 2007.

JOURNAL PUBLICATIONS:

Rachel Yudkowsky, Cristian Luciano, Pat Banerjee, Alan Schwartz, Ali Alaraj, G Michael Lemole Jr, Fady Charbel, Kelly Smith, **Silvio Rizzi**, Richard Byrne, Bernard Bendok, David Frim, "*Practice on an Augmented Reality/Haptic Simulator and Library of Virtual Brains Improves Residents' Ability to Perform a Ventriculostomy*," Simulation in Healthcare 8 (1), 25-31, 2013.

Cristian J Luciano, P Pat Banerjee, Jeffrey M Sorenson, Kevin T Foley, Sameer A Ansari, **Silvio Rizzi**, Anand V Germanwala, Leonard Kranzler, Prashant Chittiboina, Ben Z Roitberg, "*Percutaneous Spinal Fixation Simulation With Virtual Reality and Haptics*," Neurosurgery 72 (Supplement 1), A89-A96, 2013.

Ali Alaraj, Fady T Charbel, Daniel Birk, Mathew Tobin, Cristian Luciano, Pat P Banerjee, **Silvio Rizzi**, Jeff Sorenson, Kevin Foley, Konstantin Slavin, Ben Roitberg, "*Role of Cranial and Spinal Virtual and Augmented Reality Simulation Using Immersive Touch Modules in Neurosurgical Training*," Neurosurgery 72 (Supplement 1), A115-A123, 2013.

Silvio Rizzi, Cristian Luciano, Pat Banerjee, "*Comparison of Algorithms for Haptic Interaction With Isosurfaces Extracted From Volumetric Datasets*," Journal of Computing and Information Science in Engineering, 12 (2), 021004-1 -021004-10, 2012.

Arun Rakesh Yoganandan, Pat Banerjee, Cristian Luciano, **Silvio Rizzi**, "*Prototyping flexible touch screen devices using collocated haptic-graphic elastic-object deformation on the GPU*," Virtual Reality-Research Development and Applications, 16 (1), 33, 2012.

Ali Alaraj, Michael G Lemole, Joshua H Finkle, Rachel Yudkowsky, Adam Wallace, Cristian Luciano, Pat Banerjee, **Silvio Rizzi**, Fady T Charbel, "*Virtual reality training in neurosurgery: Review of current status and future applications*," Surgical Neurology International, 2, 2011.

Pat Banerjee, Cristian Luciano, **Silvio Rizzi**, "*Virtual reality simulations*," Anesthesiology Clinics, 25 (2), 337-348, 2007