UNIVERSITY OF NAVARRA

SCHOOL OF ENGINEERING

DONOSTIA-SAN SEBASTIÁN

RIGID AND DEFORMABLE COLLISION HANDLING FOR A HAPTIC NEUROSURGERY SIMULATOR

DISSERTATION
submitted for the
Degree of Doctor of Philosophy
of the University of Navarra by

Goretti Echegaray López

under the supervision of

Diego Borro Yágüez

December, 2012

SERVICIO DE PUBLICACIONES DE LA UNIVERSIDAD DE NAVARRA

ISBN 978-84-8081-347-1

Nire guraso eta anaiei

Agradecimientos

Una vez llegados a este punto, posiblemente estas lineas queden cortas para agradecer a tantísima gente que de una forma u otra me ha ayudado a sacar este proyecto adelante. Tantísimo que agradecer a tanta gente, que sinceramente no creo que llegue a transmitir todo lo que me gustaría.

En primer lugar agradezco al Gobierno de Navarra la concesión de una de sus becas doctorales, concretamente la obtenida a través del *Plan de Formación, Investigación y Desarrollo*.

Quisiera agradecer a Alejo Avello, Jordi Viñolas y Luis Matey por darme la oportunidad de realizar mis estudios de doctorado en el área de simulación del CEIT, así como al personal de administración y servicios por hacer que nuestro trabajo sea mucho más ameno.

Por supuesto, mil gracias también a quien desde un principio hizo que me sintiera como una más en este mundo paralelo. Mi director de tesis, Diego Borro. Gracias por la confianza que has depositado en mí y por el esfuerzo y las horas que todo esto haya podido suponer. Saludos, ¡programas!

En el último año de mi etapa universitaria y desde tierras alemanas, tuve la suerte de conocer a un hombre que supo estar ahí siempre que lo necesité: Alex García-Alonso. Gracias por tenderme la mano de forma totalmente altruísta, por tus sabios consejos y sobre todo por confiar en mí.

Esta etapa que cierro no hubiera sido igual sin todos los compañeros de despacho y departamento con los que día a día he compartido tantas cosas. Aunque creo que hay uno al que debo un agradecimiento especial. Imanol Herrera, mi compañero de batallas. Gracias por ese aporte técnico que tanto me ha hecho aprender, pero gracias sobre todo por haber hecho tan llevaderas esas horas interminables de trabajo y esos ”carabirubi” ante los errores incomprensibles. Gracias también por estar dispuesto a ayudar siempre, porque no me imagino

acabando este proyecto sin tu ayuda. Pero no me olvido del resto: gracias a Alba por nuestras confidencias y las cajas enormes de zapatos, a Gorka porque gracias a ti hablo con miedo a que me hagan rimas con cada frase, a Ibai por hacerme reír tanto con esas caras y comentarios inexplicables (”en gabacho”), a Aitor, con el que espero seguir compartiendo horas de risas e indignaciones (¿Esto qué es?), a Fran, frambuesa, o ”fran” de huevo. Gracias por ese aporte friki al despacho. A Alberto, un millón de gracias por tus consejos y confidencias, pero ante todo gracias por esa gran amistad que me has dado. A Carlos, mi ”time destroyer” particular. Porque espero que retomes la bonita costumbre de cantar aquello de ”buenos días gente del despacho” (sabes que nos encantaba). A Hugo, por tu arte con el Paint. Y ante todo por haber sido un apoyo fuerte durante todos estos años, sobre todo cuando en esta última etapa me quedaba sola trabajando hasta altas horas. Aunque no lo creas, tus ánimos sí servían, y mucho.

Podría extenderme agradeciendo a cada una de las personas que tanto dentro como fuera del despacho han formado parte de mi vida durante estos años. Jairo, Maite, Gaizka, Aiert, Ilaria, Ibon, Iker, Pedro, Borja, Sergio, Iñaki, Josune, Ainara B., Ainara P., Ane, Álvaro, Javi, Jorge, Tomek, Yaiza, Ainhitze, Pablo, Xabi, Jorge Juan, Emilio, Imanol P., Manolo, Txemi, Ángel, Ana Leiza y a muchos más que probablemente me esté dejando en el tintero, pero no por eso sean menos importantes.

No quisiera dejar fuera de estas lineas a mi kuadrilla Mattane, Irantzu, Unai, Ane, Niko, Miren, Lander, Lorea, Ekaitz, Idoia y Endara. La mejor kuadrilla del mundo. Gracias por haberme animado tanto y estar ahí siempre. Sois los mejores. Y por supuesto, mil gracias a Markel, una de las personas que más me ha sufrido sobre todo durante estos últimos meses de intenso trabajo. Muchas gracias por tus palabras de aliento cuando pocos han sabido ver que las necesitaba. Por confiar y creer en mí, y por estar ahí siempre que lo he necesitado.

No puedo cerrar este apartado sin mencionar a los que sin lugar a dudas han sido mi mayor apoyo desde el día en que nací. Mis padres, Fernando y Goretti. Gracias por apoyarme y ayudarme a realizar todos y cada uno de mis sueños. Por haberme dado una educación de la que estoy muy orgullosa, y por haber creído en mí siempre. Sin vuestros ánimos, vuestras palabras de tranquilidad y vuestro apoyo incondicional nada de esto hubiera sido posible. Gracias también a mis hermanos Fernando y Martín, mis dos pequeños hombres. Los que siempre han conseguido mantener a flote mi lado más infantil y de los que tan orgullosa me siento. Mil gracias a toda mi familia, esto es por y para vosotros.

Abstract

Simulation has been widely used for training and rehearsing difficult or unusual actions in several fields such as aviation and the military. However, the simulators available in some disciplines do not fulfil the requirements of reliability and accuracy that users demand. This happens in neurosurgery. In order to overcome these difficulties, this thesis presents a multimodal Neurosurgery Simulator focused on patient-specific surgical learning and training.

One of the aspects that most influences the behavioural reality of a simulator is the way in which the scene objects interfere. For that reason, detecting collisions and giving them a feasible response is particularly important. This work presents the collision handling methods for rigid and deformable volumetric objects and their haptic response to be integrated into the Neurosurgery Simulator. With the aim of evaluating our methods in terms of continuity and stability, the present document analyses the time consumption of the collision handling algorithms and the stability of the force parameters they return.

Real-time virtual reality simulators require accuracy but are also time dependent. Thus, their computational cost is a vital aspect. This thesis also proposes a methodology to optimize the time consumption of collision detection algorithms that are based on the uniform spatial partition technique. It is validated experimentally and compared to other approaches. Additionally, the optimization is applied to our deformable collision detection method in order to improve its performance.

Resumen

La simulación como herramienta de entrenamiento y ensayo ha sido extensamente utilizada en diferentes áreas tales como la aviación o el ejército. Sin embargo, los simuladores de ciertas disciplinas no cumplen con los requerimientos de fiabilidad y precisión que los usuarios demandan. Éste es el caso de la neurocirugía. Con objeto de superar estas dificultades, esta Tesis presenta un Simulador multimodal de Neurocirugía orientado a la enseñanza.

Uno de los factores que más afecta al realismo del comportamiento de un simulador es la forma en la que interfieren los objetos que componen la escena. Por esta razón, la detección y respuesta a colisiones son especialmente importantes. Este trabajo presenta métodos de detección y respuesta a colisiones para objetos volumétricos tanto rígidos como deformables y su correspondiente respuesta háptica, los cuales serán integrados en el Simulador de Neurocirugía final. Con el fin de evaluar nuestros métodos en términos de continuidad y estabilidad, el presente documento analiza el tiempo de detección y respuesta a colisiones de ambos algoritmos, así como la estabilidad de los parámetros de fuerza que devuelven.

Aparte de la precisión, los simuladores de realidad virtual en tiempo real dependen del consumo de tiempo de sus módulos. De hecho, el tiempo computacional es un factor crítico en este tipo de simuladores. Esta Tesis propone también una metodología que optimiza el consumo de tiempo de algoritmos de colisión basados en la técnica de subdivisión espacial uniforme. Se ha validado experimentalmente y comparado con otras propuestas. Además, la optimización se ha aplicado al método de detección de colisiones deformables propuesto en esta Tesis.

Contents

List of Figures

List of Tables

List of Algorithms

List of symbols

VR	Virtual Reality	3
CT	Computerized Tomography	8
MRI	Magnetic Resonance Imaging	8
$USUHS$	Uniformed Services University of the Health Sciences	10
$VMELab$	Virtual Medical Environments Laboratory	10
DoF	Degrees-of-Freedom	11
VE	Virtual Environment	14
FEM	Finite Element Method	15
BVH	Bounding Volume Hierarchy	29
$AABB$	Axis-Aligned Bounding Box	30
$k-DOP$	Discrete Orientation Polytopes	31
OBB	Oriented Bounding Box	31
CPU	Central Processing Unit	35
GPU	Graphics Processing Unit	35
VTK	Visualization Toolkit	133
$TLED$	Total Lagrangian non-linear Explicit Dynamics	139
CUN	Clínica Universitaria de Navarra	152

Formulation

$cell\_size$	Predefined size of the cell used to subdivide the space	26
ind	Indice associated to a cell	26
$numCells$	Number of cells in each axis	26
c	Centre	34
r	Radius	34
k	Stiffness constant	60
p	Penetration depth	60
$\mathbf{n}$	Contact normal	60

Part I

Introduction

Chapter 1

Introduction

The art and science of asking questions is the source of all knowledge

THOMAS BERGER

Virtual Reality (VR) simulations are computer-generated versions of real-world objects or processes presented in a 3-dimensional, multimedia format. This technology allows users to explore and manipulate environments in real time. Simulation tools are now extended to many fields such as aviation, military, architecture, entertainment and medicine. Our work is focused on VR medical simulators for learning purposes. The following section briefly introduces the reader to the world of VR Surgery Simulation and its teaching capacities.

1.1 Surgery Simulation

Medical simulation is a branch of simulation technology related to education and training in medical fields.

Medicine has gone through a substantial transition over the last decades. Due to the new medical procedures emerging every day, medical knowledge doubles every $6-8$ years. For that reason, the way in which education for health-care professionals is designed demands significant changes. Since the half-life of medical information is so short, the teaching process for students and professionals extends over years. Consequently, the need for an accurate education and training model has become essential.

These needs and the advances in educational and training technology have led

to an increase in the number of learning tools, including computer simulators.

The earliest forms of simulation for medical purposes consisted of dissections of human cadavers or animal models. However, the spread of computer capabilities created a new paradigm in simulation: the possibility of implementing new technologies to serve as simulation platforms.

This idea takes its roots from aeronautics and flight simulators. The military was the first to explore the potential of computers to train its members. Due to the success of the experience many different field experts attempted to adapt simulation technologies to their own needs. The rapid advances in hardware and the advent of computer-generated graphics in the 1980s led to the development of the sophisticated and immersive simulators that are currently in use. In the medical realm, anaesthesiology was the precursor to implementing new technologies for training purposes (Bharath, 2006).

Medical training began with the use of mannequins in the 1950s. The most notable prototype at that time was the Resusci-Anne mannequin (Laerdal, Wappingers Falls, New York, USA). It was used for Cardiopulmonary Resuscitation (CPR) training. Encouraged by Dr. Bjorn Lind and other anaesthesiologists, Laerdal created a mannequin to demonstrate mouth to mouth resuscitation in 1960.

Figure 1.1: First training mannequin: Resusci-Anne

However, the first computer controlled mannequin simulator did not appear until the mid 1960s. It was called Sime One, and it was created by the engineer S. Abrahamson and the physician J. Denson at the University of California. It realistically simulated the tasks of endotracheal intubation and the physiological responses of the patient (Denson J.S., 1969).

The earliest example of a part task medical trainer is the cardiopulmonary

patient simulator Harvey. It was developed in 1968 by M. Gordon at the University of Miami. Harvey is a full sized mannequin that performs 27 different cardiac functions of the human body, varying blood pressure, breathing, pulse, heart sounds and murmurs. Its effectiveness in medical skills teaching was sponsored by the National Heart, Lung and Blood Institute in 1987. Thus, it served as inspiration for more portable and smaller cardiology patient simulators.

Figure 1.2: Cardiopulmonary patient simulator Harvey

The development of more complete models of human physiology enabled higher fidelity simulators. The screen-based simulator SLEEPER was built with a multicompartment model of human physiology and pharmacology. Nonetheless, owing to its complex system it required huge computing power, which was not available in desktop computers. Because of their complex models and non-affordable technologies, the simulators developed until the late 1980s where aimed at a specific audience. The physician H. Schwid and the programmer Daniel O'Donnell developed the concept of screen-based simulation by simplifying the models so they could be run on a desktop computer and therefore reach a wider audience. The SLEEPER simulator, which became a broader application called BODY, was expanded and commercialised in a product called the Anaesthesia Simulator Recorder in 1989. It was later evolved into a family of screen based simulators marketed by Anesoft Corporation (Cooper and Taqueti, 2004).

VR as applied to surgery simulation was first proposed in the early 1990's (Satava, 1993) and focussed on task simulation. Owing to the increasing power of computer processing and the availability of more advanced devices such as haptics, surgery simulation gained in sophistication and realism.

VR is increasingly becoming a promising area with a great potential for improving and modifying the learning and training experience. VR training simulators can support experiential learning due to the rich, interactive and

attractive educational context they provide.

A great responsibility is put on these educational mechanisms. For that reason, in some disciplines conditions of strict fidelity and accuracy are demanded for their simulators. This is the case in neurosurgery. However, the simulators in use nowadays do not meet these requirements. In fact, neurosurgery students and residents must still attend real interventions carried out by expert neurosurgeons in order to acquire specific aptitudes.

Using VR in education and learning has clear benefits in contrast to traditional teaching methods. The most significant benefits are:

- Experiential learning: Virtual environments can provide a promising enhancement to learning through experience. Many authors, such as Manovani (Mantovani, 2001) emphasize the high potential of VR and its capacity to provide a rich, interactive and engaging educational context where students learn by doing via first-hand experience. Direct experience is vital in the learning process and education with VR simulators encourages active participation rather than passivity. Knowledge is more effectively assimilated by doing and experiencing than reading formal descriptions. Preliminary studies (Anderson, 1996) note that users' performance in understanding abstract problems is considerably greater when tracking 3D worlds where abstract entities are represented by objects.

- Access unreachable or dangerous areas: VR allows learning in contexts where in real life access would be complex or even impossible, e.g. moving inside a human body. Furthermore, activities which entail any risk of damage to the user or the patient can also be performed without danger. Additionally, VR simulators permit virtually training with expensive equipment without having to buy the machine itself.

- Increase motivation: The interaction with a VR environment can be approached like a game, which is intrinsically motivating.

- Adaptability: Simulators can be adjusted to the specific necessities of the task to be performed. What is more, owing to the different characteristics and needs of each learner, the environment, task or teaching process can be modified in order to adapt to the circumstances.

- Evaluation: Sessions in VR environments can be easily monitored and recorded with the aim of evaluating the student in one or more tasks.

- Interaction through stimuli: One of the most important keys of the teaching capacity of VR is the possibility of actively interacting with the virtual environment through devices such as a mouse or a haptic device. Our senses play an important role in the learning process, and VR has the capability of offering multiple sensorial experiences through touch, eyesight and hearing.

1.1.1 Phases of the Surgery Simulation Process

Any surgery simulation process covers various disciplines that need to be connected to get plausible and convincing results. In general terms, a surgery simulator consists of the following areas:

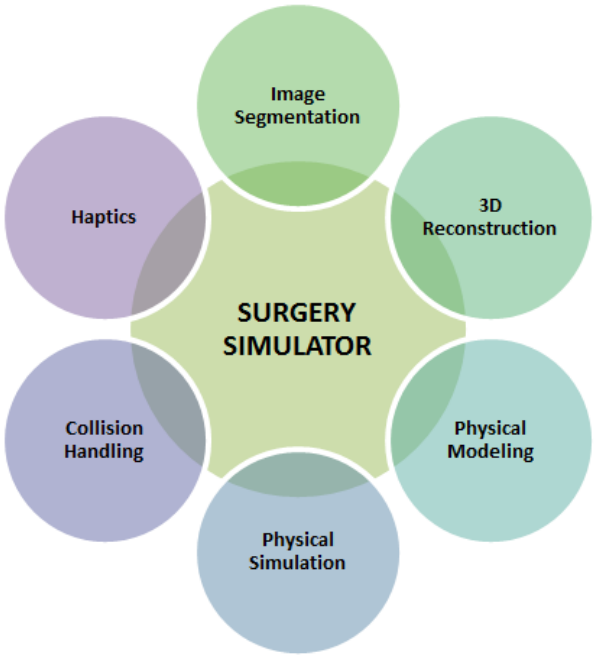


Figure 1.3: Phases of a Surgery Simulator

- Image segmentation: The goal of the image segmentation in surgery simulators is to convert the representation of medical images into something that is more meaningful and easier to analyse for the computer. It consists of partitioning a digital image into multiple segments. Segmentation can be used to locate tumours or other pathologies, measure tissue volumes, make

diagnosis, plan specific treatments or study anatomical structures.

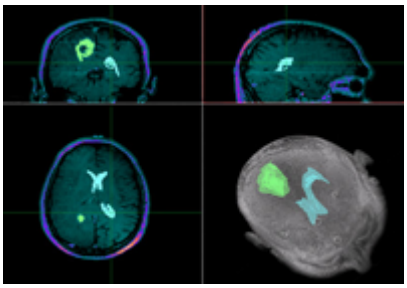


Figure 1.4: Image Segmentation in a Neurosurgery Simulator

- 3D reconstruction: Three-dimensional reconstruction is the creation of 3D models from a set of images, capturing the shape and appearance of the real object. For example, a reconstruction of Computerized Tomography (CT) scans or Magnetic Resonance Imaging (MRI) images can be created for diagnostic or therapeutic purposes.

Figure 1.5: 3D Reconstruction of a CT Scan of a minimally displaced fracture of a radial head

- Physical modelling: Different properties can be defined for each reconstructed element. Different parts of the human anatomy have different tissue properties, and each requires a different treatment when interacting with them.
- Physical simulation: With the aim of reproducing the behaviour of the elements over time a physical simulation module is required. As previously

stated, different elements can have distinct properties. For that reason, their behaviour in view of diverse situations and stimuli must also vary. The way in which the elements in the simulation behave is determined by means of this step.

- Collision handling: One of the aspects that most influences an environment's approximation to reality is the way in which the elements interfere with each other. Once interpenetrating objects are identified, constraints or other techniques restrict unwanted movements to give a solution to the interferences.

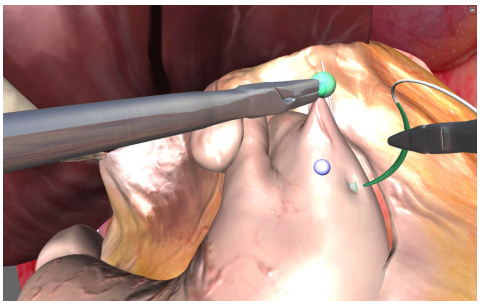


Figure 1.6: Collision handling is an essential task in any kind of simulator

- Haptics: Thanks to haptic devices users can interact with virtual objects through the sense of touch receiving force feedback as a response to collisions with the environment. Haptics significantly increase the accuracy and efficiency of the procedure in addition to providing the sensation of immersion.

Figure 1.7: Phantom Omni haptic device

The last two stages make up the concept of *"haptic rendering"*. The objective of the haptic rendering process is to stimulate the sense of tact in the users, with the aim of achieving a more realistic and interactive simulation. This thesis focuses its attention on the haptic rendering phase.

1.2 Surgery Simulators

VR surgery is currently used in many procedures and specialties, such as endoscopy (Walsh et al., 2012), microsurgery (Wen et al., 2010), neurosurgery (Lee et al., 2012a), urology (Lee et al., 2012b), orthopaedics (Zhang et al., 2011) or ophthalmology (Carvalho, 2012). With the advent of more sophisticated and more accurate tools, VR applied to surgery is gaining acceptance in more areas of the medical realm.

The world leader in the development and application of medical simulation programs is the National Capital Area Medical Simulation Center[1] or *SimCenter* which is part of the Uniformed Services University of the Health Sciences[2] (USUHS). Here medical procedures that pose a high risk to patient safety are simulated, which allows healthcare personnel to develop and maintain the cognitive and psychomotor skills needed for safely and effectively performing medical tasks.

Trying to meet the training objectives of the SimCenter, the Virtual Medical Environments Laboratory[3] (*VME Lab*) , which is part of the Surgical Simulation Laboratory, adapts computer technology for medical training through simulation. The VME Lab is active in developing new applications that include computer based imagery used in conjunction with haptic feedback devices. Their main goal is to create life-like simulated tasks and make them available to trainees on the computer screen.

Back in the early 2000's, a company called *ImmersiveTouch*[4] was created in Chicago from work done as part of an advanced technology program at the National Institute of Standards and Technology[5]. Over the years the company worked with beta customers to develop its products. Currently they offer

[1] http://simcen.usuhs.edu/Pages/default.aspx
[2] http://www.usuhs.mil/
[3] http://www.simcen.org/
[4] http://www.immersivetouch.com/
[5] http://www.nist.gov/index.html

augmented virtual reality systems with 3D graphics and haptic feedback. Among their projects, the most notable ones are the cranial simulations oriented toward ventriculostomies, spinal simulations and anatomical explorations. A cataract simulator is also under development. They offer a simulation platform that can be used as the base of many simulators (Luciano et al., 2005), (Lemole et al., 2007).

Our work is focused on craniotomy and neurosurgery procedures. The following sections give some examples of the craniotomy and neurosurgery simulators that have been developed over the last decades.

1.2.1 Craniotomy and Bone Drilling Simulators

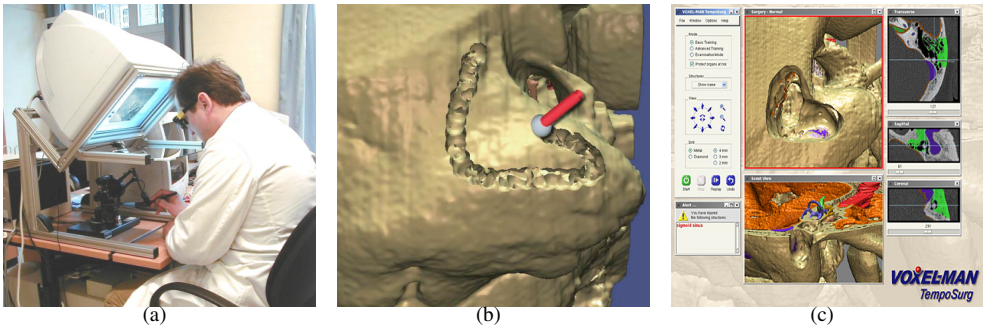
In 2002, a group at the University of Hamburg developed a virtual petrous bone surgery simulator for specific laterobasal approaches (Pflesser et al., 2002). This application allows the simulation of specific surgical approaches and cuts with high quality visualization, which is achieved by what at that time was a new multi-volume visualization technique. The user can learn about the three-dimensional anatomy of the temporal bone from the viewpoint of a real otosurgical procedure using a Sensable 3 DoF (Degrees-of-Freedom) Phantom device.

The same group also presented a temporal bone surgery simulator called TempoSurg (Petersik et al., 2002). It provides tactile haptic feedback and incorporates a foot pedal to control the speed of the virtual drill.

Another example of a real-time temporal bone simulator is the Visible Ear Simulator (Figure 1.9). It is oriented toward ear surgery procedures, in which a surgeon drills into the temporal bone to gain access to the middle or inner ear. Its main goal is to support the development of anatomical insight and training both students and experienced otologists in drilling skills. A physically plausible drilling experience is achieved by means of a Phantom Omni force feedback device.

The Stanford Bio-Robotics Lab[6] also conducts research in user-friendly robotics and surgical robotics. The physical interaction between human beings and computer-driven actuators is one of their main lines of research. Their projects centre on haptics, surgical simulation and robotic design, among other areas. In the surgical simulation realm, they primarily focus on bone drilling simulators

[6] http://biorobotics.stanford.edu/



(a) (b) (c)

Figure 1.8: (a) Petrous bone surgery simulator working environment, (b) Petrous bone cutting (Pflesser et al., 2002), (c) The temporal bone simulator TempoSurg by the University of Hamburg (Petersik et al., 2002)

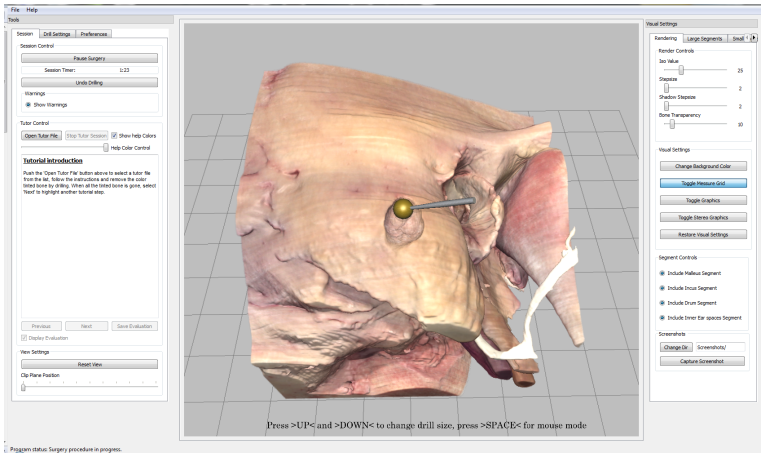


Figure 1.9: The Visible Ear Simulator while drilling the temporal bone

(Morris et al., 2006), (Morris et al., 2005).

He and Chen (He and Chen, 2006) proposed a freehand-controlled bone drilling simulation framework with a prototype simulation system for training skull-bone drilling procedures (Figure 1.10). One of the biggest contributions they made is the modelling of torque rendering. The haptic device used is a Sensable 6 DoF called Phantom Premium. The user can drill a virtual skull bone with multi-sensory feedback such as force, torque, vision and sound.

Figure 1.10: Freehand-controlled bone drilling simulator including a Sensable 6 DoF Phantom Premium device (He and Chen, 2006)

One of the training tools that the VME Lab is actively developing nowadays is the Intracranial Hematoma Craniotomy simulator (Acosta et al., 2007), which helps the learner to practice the skills required to perform a craniotomy, including scalp cutting and retraction. Typically used bone cutting tools, which have been created from real instruments, are modelled and controlled by a haptic device.

Kockro and Hwang (Kockro and Hwang, 2009) created a virtual model of the temporal bone called Virtual Temporal Bone. Together with an environment called Dextroscope which will be discussed further in 1.2.2, the virtual model can simulate several cranial base surgical procedures. Furthermore, they calibrated a Phantom arm with a bone drilling application in order to experiment with tactile haptic interaction.

The medical education and training company SimQuest LCC [7] created a burr hole simulator to train users to drill a burr hole into the skull for diagnostic purposes or therapeutic decompression. It enables non-neurosurgeons to learn and practice the procedures before operating on real patients. It uses a Hudson Brace drill handle attached to a 6 DoF haptic interface device that integrates two modified Falcon 6 DoF haptic force feedback systems (Figure 1.11). This design allows users to stand while holding the real drill, which is attached to the mentioned force-feedback devices, just as they would in an operating room. It allows nearly the same movements and feeling of forces and torques as produced in the actual procedure. Results of an assessment study indicated that spending 20 minutes on this simulator was likely to have measurable benefits for less experienced individuals performing their first craniotomy procedure.

[7] http://www.simquest.com/

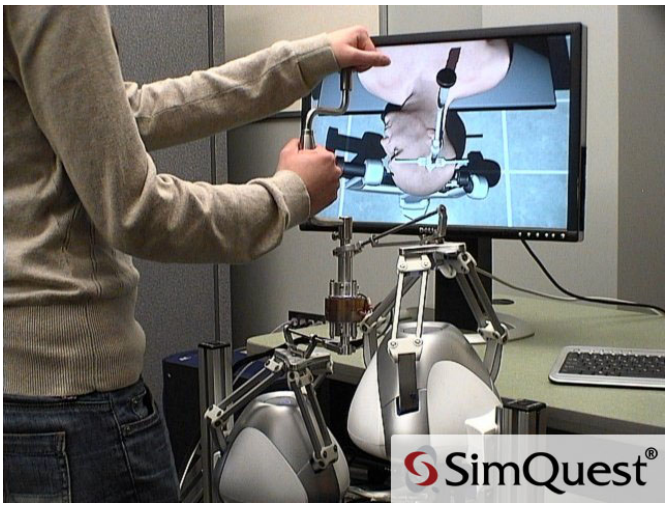


Figure 1.11: SimQuest LCC Burr Hole Simulator

1.2.2 Neurosurgery Simulators

In the last decades many VR platforms have been designed and implemented to support education and training in different areas of learning. In the realm of neurosurgery, experts are increasingly demanding simulation mechanisms that facilitate and guide their procedures. Thus, the management of neurosurgical disorders has gone under significant advances in the last years. This has led to a significant growth in the amount of successfully resolved neurological diseases.

For example, Giorgi et al. (Giorgi et al., 1994) used a Virtual Reality system to guide a mechanical toolholder in a space of stereotactic neuroanatomical images. High precision encoders on the arm are attached to the stereotactic frame. It helps the surgeon control and check the orientation of different approach trajectories.

Z. Wang et al. (Wang et al., 2000) also presented a system for stereotactic neurosurgical planning and support. A 3D model of the interior structure of the patient's brain is reconstructed and displayed. The patient's head and the brain model are mapped using marker registration. A robot arm locates the predefined incision point and the orientation of the incision route on the patient's head. Then the surgeon can insert a medical instrument into the patient's head and the surgery is performed successfully. With the help of a virtual environment (VE), this system could also be used for teaching and training.

P. Wang et al. (Wang et al., 2007) (Wang et al., 2006) presented a virtual-reality neurosurgical simulator which incorporates simulation of surgical prodding, pulling and cutting. The simulator uses boundary element technology to develop real-time realistic deformable models of the brain. 3D stereo-vision is implemented and two hand-held force-feedback devices are used.

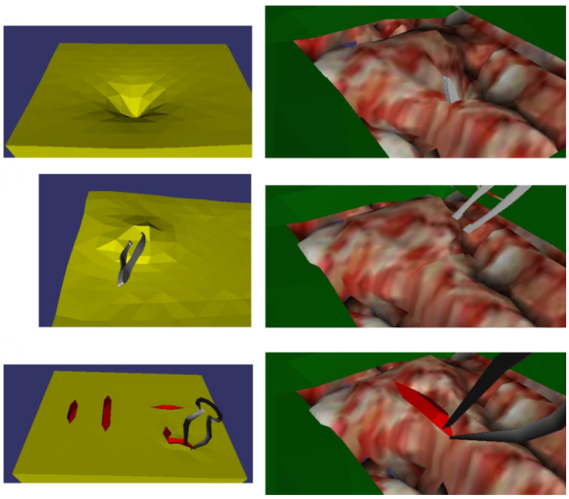


Figure 1.12: Prodding, pinching and cutting examples in (Wang et al., 2007)

A team of experts from the NRC research institutes in Canada proposed the NeuroTouch Simulation[8] platform (Figure 1.13), which comprises neurosurgical simulation training for craniotomy and endoscopy. The craniotomy procedures include microsurgery tasks such as tumour debulking and bipolar cautery. The endoscopy procedure enables endoscopic third ventriculostomy and trans-nasal navigation. Photorealistic 3D rendering of tissues and bleeding is offered, as well as a highly delicate force feedback on surgical tools. The deformation and rupture of soft tissues is determined by the Finite Element Method (FEM).

As mentioned in the explanation of the ImmersiveTouch platform, augmented reality has also been implemented for surgical purposes. Gleason et al. (Gleason et al., 1994) combined three-dimensional computer-reconstructed neuroimages with a novel video registration technique for virtual reality-based, image-guided surgery of the brain and spine. This technique superimposes a 3D reconstructed MR or CT scan on a live video image of the patient using augmented reality. This

[8] http://www.neurotouch.ca/eng/simulators.html

Figure 1.13: Neurotouch Simulation platform

technique allows the surgeon to localize cerebral and spinal lesions. The scan is segmented into the relevant components such as tumours, oedema, ventricles, arteries, brain and skin, and the surgeon can determine the optimal surgical approach. The surgeon's proposed approach is displayed in the operating room at the time of surgery using a portable workstation. Augmented reality is also playing an important role in neurosurgical endoscopy.

With regard to the 3D neurosurgical planning tools, the most significant advance came with the arrival of the Dextroscope simulation environment in the 1990s. This uses CT and MRI images in order to create specific volumetric models. It takes advantage of its stereoscopic imaging capacity, and also uses positional controllers to manipulate models with natural movements. However, this mechanism does not have haptic feedback. The Dextroscope environment has been used for many tasks, including surgical planning. For instance, a platform called VizDexter (Kockro et al., 2007) provides tools to coregister data, perform segmentations manually, make measurements and simulate multiple intraoperative viewpoints using Dextroscope. It has also been used for head positioning, craniotomy and virtual aneurysm clipping simulation through a procedural application created by Wong et al. (Wong et al., 2007).

1.3 Motivation

Simulation has been widely used for training and rehearsing difficult or unusual actions in several fields. Indeed, the use of simulators has become mandatory in many training programs. However, the simulators available in some disciplines do not fulfil the requirements of reliability and accuracy that users demand. This happens in neurosurgery. In fact, every year neurosurgery students from all over Europe must attend real interventions carried out by expert neurosurgeons in order to learn specific procedures.

One of the most important goals of training simulators nowadays is for trainees to assimilate the skills needed to execute complex surgical procedures before practicing them on patients. Of the various characteristics of the environment that make it possible to meet that objective, two key features are: immersion and presence. Immersion is described as the experience of being surrounded by a virtual world. Presence, in turn, is a multidimensional perception that reports a feeling of not just being surrounded by the virtual environment but also being part of the virtual world, at the same time that other beings also exist in that VE (Schuemie et al., 2001). Receiving realistic haptic and visual stimuli strongly conditions the immersion and presence of the application.

As stated previously, accuracy is a critical condition when working with neurosurgery simulators. On account of the existing limitations in haptic interfaces, such as volume-rendering techniques or physically based algorithms, much work is being done in this realm, and many promising advances have been made throughout recent years (McNeely et al., 1999).

In order to overcome these difficulties, our goal is to develop a multimodal Neurosurgery Simulator focused on patient-specific surgical training of brain tumour resection. Different areas need to be combined in order to construct a feasible and completely accurate tool. Therefore, the disciplines we will integrate are: medical imaging, 3D geometrical reconstruction of tissues, real time volume rendering, physical modelling, simulation, collision handling, visualization and haptics.

Having a realistic simulator which accurately describes the behaviour of the tissues will offer clear benefits over other simulators. The first one is that the simulation of an accurate model will be useful for new surgical equipment design since dimensions, working angles and shapes can be taken into account. Additionally, this realism will be essential for surgical anatomy teaching and

training for students, residents and specialists. Finally, this type of simulators also facilitates the development of applications that analyse the accessibility and suitability of medical instrumentation.

This project can become a platform for training students in real surgery procedures. Students will get experience in the tasks involved in these operations, thereby considerably reducing the number of hours they need to spend attending long interventions. Moreover, the training of students, residents and surgeons will be done based on a realistic neurosurgery simulator with accurate patient-specific models.

1.4 Objectives

The main goal of this work is the generation of a realistic and immersive haptic and visual response to the user's interaction with rigid and deformable bodies in a neurosurgery simulator. In order to obtain a convincing response, an adequate collision detection algorithm that efficiently detects the interferences in the environment and offers accurate information about collisions must be run. All these matters are included in the haptic rendering process.

This thesis is centred on obtaining a convincing collision detection and response for a Neurosurgery Simulator oriented toward brain tumour resection. This will serve as a learning platform for students and residents. Additionally, patient-specific data will be used to model the objects, so it will offer the surgeons the possibility of rehearsing prior to a real intervention. The multimodal Neurosurgery Simulator will be split into three main phases:

- Craniotomy Simulator: This first prototype will simulate skull drilling using a haptic device that governs a virtual milling tool. The system will provide the surgeon with the visual and force feedback corresponding to an actual craniotomy intervention.

- Brain Physical Haptic Simulator: This simulator operates on an accurate model of the brain tissue. The surgeon will be able to interact with the virtual brain by means of a haptic tool. It will offer force feedback to the user via the haptic device. In addition, the deformations of the brain model due to the impact of the surgical tool will be emulated, giving a realistic idea of the physical behaviour of the tissue.

- Integration: This consists of the combination of both the Craniotomy Simulator and the Brain Physical Haptic Simulator. It will be made up of a skull model obtained from real CT data and a model of the brain tissue inside it. The brain model is acquired from MRI data from the same patient. The surgeon will be able to drill the skull and interact with the virtual brain with a haptic tool. Further, the surgeon will be warned whenever the milling tool touches the brain tissue while drilling the skull. Once the drilling task is accomplished, the tool will be changed and the new task will be enabled.

An additional fourth stage would involve the expansion of our simulator to a final multimodal Neurosurgery Simulator. In addition to the features mentioned above, it would include the option of cutting the brain tissue in order to let the surgeon access the tumour and resect it. The resulting multimodal simulator would enable surgeons to practice actual operations.

For the first stage we will focus on the three key elements needed to obtain the required realism: refined visual representation, realistic collision handling and haptic interaction.

The second stage involves the design and implementation of a Brain Haptic Physical Simulator. In this case, in addition to the components needed for the Craniotomy Simulator, accurate rendering of physical behaviour, including deformable collision handling and physical simulation, is required.

The visual aspect and realism of the simulator greatly affects its ability to serve as an effective teaching tool. Therefore, we will use detailed medical images to model the skull. CT data will be acquired since it lets bone be rendered more clearly. The brain model will be loaded from a real MRI image of the patient. The use of patient-specific images gives the simulation systems the ability to train, as data from different anatomies and diseases can be used. This widens the variety of training scenarios and, consequently, increases the range of knowledge that can be acquired. Due to this characteristic the simulator will be used for rehearsal in addition to training.

The haptic interaction and the behaviour of the model are deeply intertwined since the haptic response depends on the physical behaviour. The Neurosurgery Simulator will offer a wide variety of possible skills for training or rehearsal, such as the friction of the tool on the skull. The collision handling algorithm will give a realistic sensation of working with a real milling tool, together with the visual feedback and the haptic interaction.

One of the advantages of the proposed simulators is that the input data

correspond to patient-specific medical images. In contrast, traditional training simulators use generic data and organ atlases. This project will introduce an integrated solution to facilitate surgery rehearsal and the execution of the intervention.

1.5 Dissertation organization

This dissertation is divided into 7 chapters. Chapter 1 has introduced the roots and evolution of surgery simulation, as well as the current objectives it covers. In addition, it has introduced some of the surgery simulators developed during recent years. The factors that have motivated our work and the main objectives we wanted to reach have also been presented.

As this thesis is focused on collision handling for a neurosurgery simulator, Chapter 2 introduces some basic collision detection and response techniques used in the literature. These bases will be useful for understanding the subsequent presentation of a variety of collision methods that have been implemented recently for different surgery simulators.

Chapters 3 and 4 present the collision handling algorithms developed in this thesis. The first one centres on rigid volumetric object collision detection and response in order to integrate them into a Craniotomy Simulator framework. The second one expounds the collision detection and response methods for deformable bodies that we have implemented as part of a Brain Haptic Physical Simulator. The experiments performed to analyse both collision handling methods are also presented at the end of each chapter.

Due to an optimization gap found in the literature concerning collision detection techniques based on Uniform Spatial Partitioning, we have also developed a methodology that faces the problem. This work, together with the obtained results, is explained in Chapter 5.

As previously mentioned, the developed collision detection and response techniques have been oriented in such a way that they can be integrated into two surgery simulators: a Craniotomy Simulator and a Brain Haptic Physical Simulator. Both are described in Chapter 6. Additionally, these simulators have been combined to form a Haptic Physical Neurosurgery Simulator, which is also presented in the same section.

Finally, Chapter 7 discusses the conclusions of this thesis and possible future

lines of work.

Chapter 2

State of the art

As previously mentioned, surgery simulators cover various areas which combined form accurate and convincing training tools. Haptic rendering is one of them. This thesis is centred on the haptic rendering field, which is responsible for computing the necessary forces to feel a realistic interaction between the virtual representation of the haptic device and the virtual objects of the scene. Therefore it assures that a correct force is transmitted to the user through the haptic device (Salisbury et al., 2004).

Haptic rendering algorithms consist of three basic components: collision detection, collision response and haptic control (see Figure 2.1).

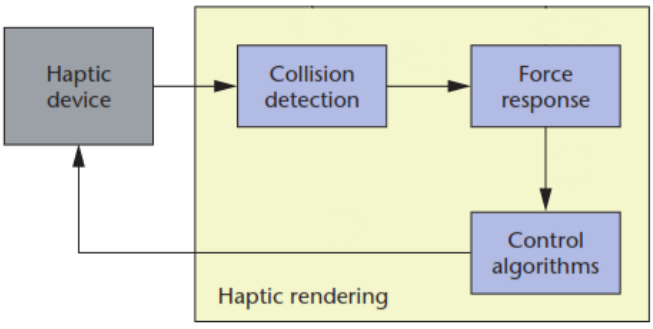


Figure 2.1: Haptic rendering is splitted into three main blocks: collision detection, collision response and control algorithms (Salisbury et al., 2004)

The collision detection module receives the position of the haptic device and the environment's information and detects interferences between the scene objects. The collision response module takes that information and calculates the

ideal force to be applied to the virtual tool and environment. Finally, control algorithms return a force to the user adapted to the characteristics of the haptic device.

Our main goal is focused on the correct collision detection and response handling of a Neurosurgery Simulator. It comprehends the collision handling for rigid volumetric objects (skull) and the one for deformable bodies (brain). Due to the presence of a haptic device in our simulator, collisions must be adapted to be combined with the haptic control system.

This section will introduce the reader into the basics of collision detection and response techniques and structures. Afterwards, collision handling methods used in different surgery simulators will be presented.

2.1 Collision Detection

The realism of an environment is strongly conditioned by the way in which the scene objects interfere each other. Different objects should not share a point in the space at the same time, so a solution must be given to the movements which place several objects at the same location at a time. A priori the objects in the virtual world are free of penetrating each other. The detection of geometric interferences between virtual bodies and the nature of the contact are the job of collision detection algorithms.

The detection of the interfering objects is not enough to later estimate a physical response to the collision. Additional information as the penetration depth and contact normals is needed. Several techniques have been implemented in order to solve this matter (Lin and Manocha, 2003). Each method is adapted to the environment's necessities, conditions and applications.

The most common methods are based on Bounding Volume Hierarchies (Biesel and Gross, 2000) (Madera et al., 2010) and Spatial Subdivision (Gissler et al., 2009b). However, many other approaches like Image-Space Techniques (Heidelberger et al., 2003), Stochastic Methods (Teschner et al., 2004a) or Distance Fields (Gissler et al., 2009a) can be found in the bibliography. The latter is not very appropriate to use in real-time for geometrically complex objects. This section will present two of the most significant techniques: Spatial Subdivision and Bounding Volume Hierarchies.

2.1.1 Spatial Subdivision

The idea of the Spatial Subdivision or Spatial Partitioning is to divide the space into convex regions called *cells*. In such a way, the zones where the objects are far away from each other are directly discarded. The attention is centred in the zones where the objects are close, to later calculate the exact intersection between them.

The Spatial Subdivision technique partitions the space into smaller volumes which are classified with identifiers (calculated according to the coordinates of the cell). Each cell has a list of the object primitives contained inside it, thus the interacting objects are classified and relations between them can be identified.

The space can be partitioned in different ways. There are many structures to be used for space partitioning, such as Voxel Grids (Teschner et al., 2003), Octrees (Garcia and Corre, 1989), K-d Trees or Binary Space Partitioning Trees (BSP Trees) (Ar et al., 2002). The following lines give a brief idea of the nature and characteristics of each one.

Voxel Grids: This technique uses uniform, rectangular, axis-aligned cells called *voxels*. The structure is based on a three-dimensional array of cells.

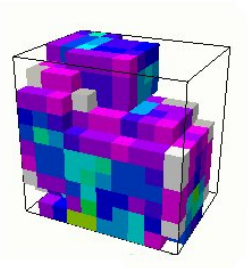


Figure 2.2: Uniform Voxel Grid structure

Voxel grids are object independent, as they can be constructed according to parameters which have nothing in common with primitives. Furthermore, the cell access is made in constant time (O(1)), since the calculation of the cell is made in constant time.

For instance, given a point with the coordinates (x,y,z), the indices ijk of the corresponding cell could be:

$$cell_i = floor(x)/cell\_size_x$$

$$cell_j = floor(y)/cell\_size_y$$

$$cell_k = floor(z)/cell\_size_z$$

where $floor(n)$ is the function which returns the largest integer not greater than n, and $cell\_size$ represents the size of the voxels in each axis. Even if this size should be equal for the three axis as the Voxel Grids should contain rectangular cells, some authors (Garcia-Alonso et al., 1994) propose parallelepiped voxel grids.

Once the indices for each cell are known, a hashing function locates them in the hash vector:

$$ind = cell_k \times numCells_x \times numCells_y + cell_j \times numCells_x + cell_i$$

being *ind* the indice of the cell in the hash vector and *numCells* the number of cells of the grid for each axis.

The chosen hashing function can change the performance of the method. Values given by this function should be uniformly distributed to get a good performance. The hash table size also influences, as the use of big ones reduces the possibility of placing several different cells to the same position. Therefore, the algorithm usually works faster. On the other hand, performance decreases due to the higher memory consumption. Some studies have been made on this topic (Teschner et al., 2004b), which have found that if the hash table is significantly larger than the number of primitives, the risk of hash collisions is minimal. However, hash functions work most efficiently if the hash table size is a prime number (Smith et al., 1995).

Concerning to the benefits of using a voxel grid, the low storage usage and fast cell access need to be mentioned. However, the biggest drawback of this structure is the fact that performance depends on the density of objects in the space being uniform. If the objects are very close to each other and clustered, a few cells have most of the objects and the majority of cells are empty. In such cases a grid is not very useful.

Octrees and k-d Trees: Octrees and k-d Trees also partition the space into rectangular, axis-aligned cells. But in these cases the cells are not necessarily uniform. The partition is made by structuring the space hierarchically: The root node corresponds to the whole space, and internal nodes represent a subdivision of

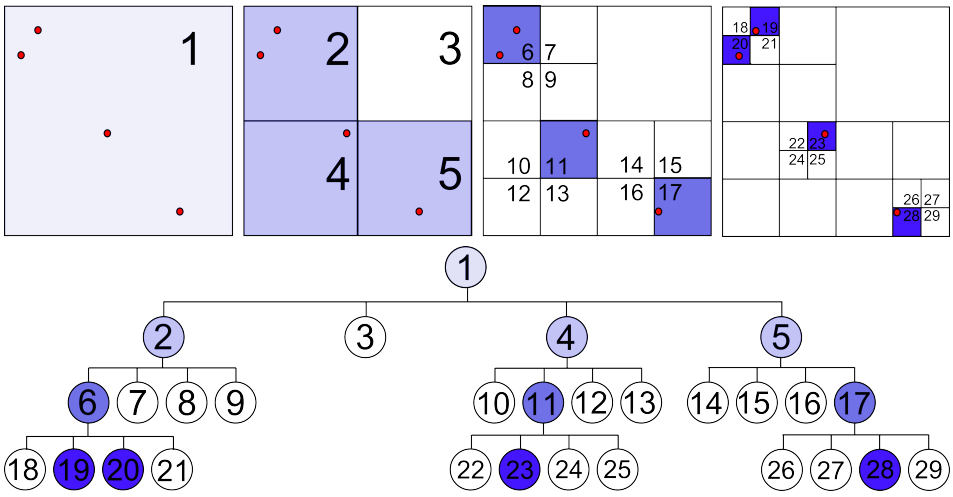


Figure 2.3: Octree structure

the corresponding region into smaller regions. Leaf nodes are cells which contain primitive lists (Figure 2.3).

In conclusion, these techniques are adaptive to local distribution of the objects. Cells are larger in zones with low density of primitives than in greatly populated areas. Therefore the updating process is made dynamically, as cells with many primitives can be subdivided while the ones with fewer primitives can be merged.

The difference between Octrees and k-d trees is the way the regions are subdivided. In 3D, an Octree divides each region into eight uniform parts along the three axes. On the other hand, k-d trees subdivide each region into two parts along an arbitrary axis (Figure 2.4).

These two methods are popular due to their relatively little storage usage and their ability to adapt to different densities. Regions that are densely populated by objects can be more finely subdivided, while sparsely populated regions can be subjected to less subdivision.

BSP Trees: The Binary Space Partitioning Tree is another hierarchical structure which divides the space into convex cells by arbitrarily oriented planes. The idea of BSP Trees was firstly formalized by H. Fuchs in 1980 (Fuchs et al., 1980).

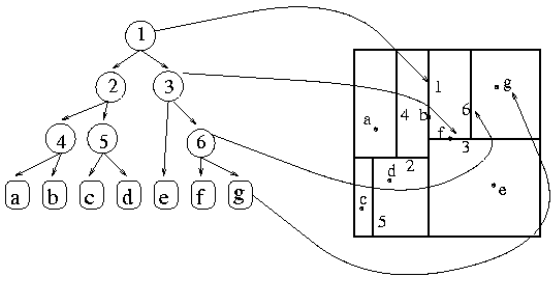


Figure 2.4: K-D Tree structure

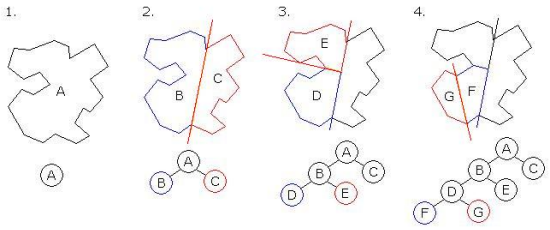


Figure 2.5: BSP Tree structure

This technique can be understood as a generalized k-d tree. There are also discrete-orientation BSP trees (DOBSP trees), in which the orientations of the partitioning planes are chosen from a finite set of orientations.

Figure 2.5 illustrates a BSP Tree structure. In n dimensional space, it is subdivided into two subspaces by a $n-1$ dimensional plane. Afterwards elements are classified as belonging to one or the other subspace. One subspace is regarded as positive and the other as negative, while the actual splitting plane is arbitrarily assigned to either space.

1. A is the root of the tree and the entire space of interest
2. A is split into B and C
3. B is split into D and E
4. D is split into F and G, which are convex and thus become leaves on the tree

One of the main problems of this type of structures is the choice of the most

efficient partitioning plane, as it strongly affects the size and performance of the tree.

- **Collision Query:**

As mentioned before, in this technique the space is subdivided into cells, which are introduced into a hash vector depending on their position in the world.

The collision detection technique with this method involves classifying all the primitives of one object with respect to the cell in which they are placed. Their position is discretized with respect to a user-defined cell size. Then the defined hash function locates the discretized position into a hash vector. So after that, each cell has a list of references to primitives contained in it.

Afterwards, primitives of other objects in the scene are discretized using the same criterion. For each discretized primitive, it is checked for intersection with the list of primitives contained in the same cell. The intersection test varies depending on the primitives.

In the case of deformable objects, the lists of primitives occurring in a cell need to be updated for every object transformation. However, there are some methods that instead of updating the whole hash table, just update the indices that actually need to be updated by using time stamps to identify them (Teschner et al., 2003). There is also another technique called *Collision Interest Matrix*, which contains flags that indicate whether collision detection has to be carried out between any given pair of objects (Garcia-Alonso et al., 1994). As shown before, by means of this structure many primitive pairs can be rejected from intersection testing.

2.1.2 Bounding Volume Hierarchies

Bounding Volume Hierarchies (BVH) are tree structures on a set of geometric objects (the data objects). Each primitive object is stored in a leaf of the tree. Leaf primitives are then grouped and enclosed within bounding volumes and stored in upper nodes. These, recursively, are also grouped within larger bounding volumes, resulting in a tree structure with a single bounding volume at the top of the tree.

Several data structures can be used to enclose the geometric objects. The choice of the structure type determines a variety of factors: the computational cost of computing the bounding volume, the updating cost, the cost of determining intersections, and the desired precision of the intersection test.

A bounding volume is a primitive shape that comprises the model and fits it as tightly as possible. They are used to approximate a collision detection quickly, as using bounding volumes to determine whether two models are intersecting or not we skip checking all primitives. In consequence, it should be computationally much cheaper than tests for the enclosed models. What is more, they are represented by a small amount of storage (or at least smaller than the storage used by the model itself), and the cost for computing such a volume for a model should be low.

Bounding volumes allow us to know whether two objects could interfere, so the fast rejection is one of the merits of these structures. For that reason, they should fit the object as tightly as possible to reduce the probability of a query object intersecting the volume but not the object. They also require efficient overlap tests and memory efficiency. If recomputation of a bounding volume is required, efficient computation would be necessary.

The following lines introduce different types of bounding volumes:

Bounding Spheres: A sphere is a very simple volume type, which only needs the centre and the radius to be represented. Spheres are fairly used due to their simplicity and the fast intersection test they allow. Moreover, they do not need to be recalculated when rotations are made, which makes them more popular in dynamic environments. Nonetheless, they are not very tightly fitting volume types for many shapes.

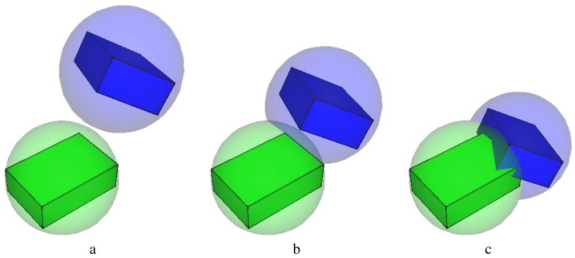


Figure 2.6: Two objects surrounded by their respective Bounding Spheres colliding

Axis-Aligned Bounding Boxes: Axis-Aligned Bounding Boxes are commonly known as AABBs.

They are even more widely used than spheres, since their test for intersection is really fast and simple. AABBs are created based on the three coordinate axis, and in contrast to spheres, they must be recomputed when objects rotate. There

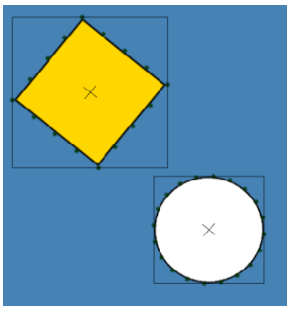


Figure 2.7: Axis-Aligned Bounding Boxes

are many ways of representing this type of volume: One of them is just storing the maximum and minimum values of the box, (the top-right and bottom-left points). There is another option which stores the centre and radius of the box. Computing the smallest AABB for an object can be done quickly by just projecting its vertices onto the coordinate axes and store the minimum and maximum values of each one.

k-DOPs: k-DOPs (Discrete Orientation Polytopes) are convex polytopes whose facets are determined by a finite fixed number of orientations. They are similar to AABBs, but their axes are not reduced to the coordinate axes, the finite number $k/2$ describes the number of directions of its halfspaces.

They are represented by its directions and the $k/2$ pairs of minimum and maximum values. Larger k -s are more flexible.

As AABBs, this type of bounding volume needs to be changed if the object suffers a rotation, and the smallest k-DOP for an object is computed in the same way by using its projections onto the $k/2$ axes.

Oriented Bounding Boxes: Oriented Bounding Boxes (OBBs) are the most tight-fitting volume for many shapes. However, their storage and testing cost are bigger than others. Their principal axes are not fixed and they move according to objects transformations.

It is hard to find the smallest OBB of an object, as an inhomogeneous vertex distribution can cause bad fitting OBBs. For that reason heuristics are applied to get reasonably tight-fitting boxes.

Apart from these four data structures, there are some more as ellipsoids, convex hulls or swept-sphere volumes (SSVs), which are less efficient and are used in less cases.

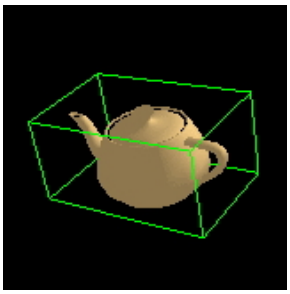


Figure 2.8: Oriented Bounding Box of a tea cup

Once the different data structures that can be used for solving the collision detection problem, some information about the way in which collisions can be found will be given in the following paragraphs.

Once the bounding volume type is decided, a hierarchy of the chosen boxes needs to be constructed to determine the collisions. A bounding volume hierarchy is basically a tree of bounding volumes, in which primitives are stored in the leaves. Each node contains a bounding box that encloses all bounding volumes or primitives of its children.

- **Building and updating process:**

There are several ways of building bounding volume hierarchies. The most used one is called top-down, which starts building the tree from the root and ends with the leaves. At each recursion step this method computes the smallest bounding box of the set of primitives, and it is splitted by ordering the primitives with respect to a well-chosen partitioning plane.

There is another building method called bottom-up, which starts with primitives and ends when inserting the root. First of all, bounding volumes for the primitives are created, and then other levels are constructed by recursively grouping primitives or bounding volumes and making new bounding volumes for them. In case of a single bounding volume, the tree is complete.

One of the facts that makes the hierarchies slow down in the case of deformable objects is the constant need of updating the tree for every change in the scenario. The hierarchy must be updated every simulation step, so the execution slows down pretty much. However, there is a method proposed by Larsson and Akenine-Möller (Larsson and Akenine-Möller, 2003) called Hybrid Hierarchy Update, which makes the update faster in some cases: This technique updates

the tree from the level n/2 to the root by the bottom-up, and the rest of the tree is updated as needed only if necessary. With this method, the change of the whole tree is not always necessary and that can strongly decrease the computing time.

Several studies have been made to decrease the time of updating the tree for every time step. There is also another technique proposed by Gino Van Den Bergen (van den Bergen, 1998) which can be used for AABB Trees, that allows them to be refitted in a bottom-up manner. This method builds the tree by allocating the leaves and internal nodes as contiguous arrays of nodes, such that each internal child node's index number in the array is greater than its parent's index. So in this way, the array of internal nodes is iterated in reversed order to refit internal nodes. This technique refits the whole tree in linear time in the number of nodes.

- **Overlapping test:**

The idea of hierarchies is to speed-up the collision detection, preventing from unnecessary tests and simplifying the query. Starting from the root node, if the bounding boxes of its children do not overlap, collision test finishes. If they do, children for both of them are checked for intersection. Continuing with this procedure, a collision will be happening if the leaves are reached and they overlap. Thus, three collision detection tests need to be implemented: The first one checks two bounding volumes for intersection. This can be seen in Figure 2.9, where two AABBs are analysed. This test is not enough to find real intersections between objects, but it is used because of its fast computation and as a previous rejection of possible collisions. When two bounding boxes collide, a more detailed test of the corresponding objects is necessary, but if they do not, the possibility of penetration can be rejected.

This thesis uses many AABBs. The intersection between two of them can be detected as in Equation 2.1:

Two AABBs do not overlap in 3D if

$$\begin{gathered} \left| (c_1 - c_2) \bullet \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right| (rx_1 + rx_2) \\ \wedge \\ \left| (c_1 - c_2) \bullet \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right| (ry_1 + ry_2) \end{gathered} \tag{2.1}$$

$$\left| (c_1 - c_2) \bullet \overset{\wedge}{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}} \right| (rz_1 + rz_2)$$

where c_1 and c_2 are the centre of each bounding box and rx, ry and rz are the radius in each axis. Figure 2.9 illustrates the intersection query.

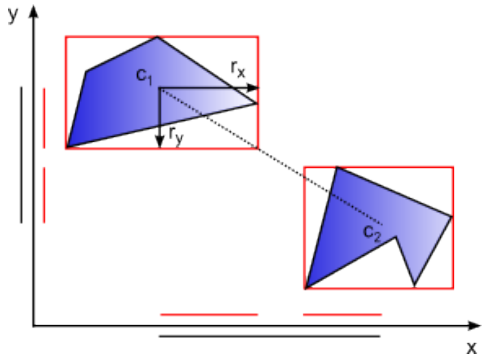


Figure 2.9: Intersection test between two objects using their AABBs in 2D

A second type of test checks intersections between a bounding volume and a primitive. This one varies depending on the primitives used. This thesis checks collisions for tetrahedra, and in the case of having an AABB against a tetrahedron the intersection test can be made by checking whether any edge of the bounding box intersects the triangles of the tetrahedron or not. All the edges of the tetrahedron are tested in the same way with respect to the squares of the box. However, there are two special cases: a bounding box occurring completely inside a tetrahedron, and a tetrahedron which is completely inside a bounding box. These two cases should be tested separately.

The last test checks whether two primitives collide or not. In the case of tetrahedra, this test is made in the same way as the previous one, but replacing the bounding box with another tetrahedron.

In this method there is an improved object approximation at higher levels of the tree. The localization of regions with collisions is fast, nevertheless the generation of the tree can be expensive. For that reason it is more used for rigid models when they can be pre-computed.

Both the Spatial Subdivision and the BVH technique do all their calculations

using the CPU. However, the use of the GPU is rapidly increasing for these matters in the last years. There are different collision detection techniques that benefit from the graphics hardware and its parallelization capacity in order to accelerate their calculus (Heidelberger et al., 2003), (Govindaraju et al., 2005), (Wong and Baciu, 2005).

2.2 Collision Response

Virtual objects do not have the properties inherent to real objects which preclude two solid bodies sharing a point in the space at the same time. The realism of a simulation is strongly conditioned by the similarity of the objects' behaviour in the real and virtual world. For that reason, a correct collision response is crucial to avoid the virtual objects penetrating each other.

Once a collision between the scene objects is detected, the collision response algorithm uses the obtained information to give a realistic physical response to the interference. The response varies depending on the application's domain: for the case of haptic surgery simulators, the collision response consists of computing the force feedback to be sent to the user via the haptic device. We must calculate an appropriate force to be applied to the haptic device in order to keep its virtual representation on the surface of the colliding object. Moreover, the colliding virtual bodies must also behave according to the forces generated by the interference. For these issues, collision response algorithms use the environmental information as the position of the colliding objects, the position of the virtual tool or other information obtained from the collision detection module to respond with a feasible feedback.

One of the most popular collision response techniques is the so-called *penalty method*. It introduces forces to the haptic interface in order to meet a constraint, that is, it generates a reaction force if a constraint is not met. Its effect is directly proportional to the penetration of the virtual haptic point into the model. In other words, the reaction force is a direct function of the haptic position inside the virtual model.

When the virtual objects have complex geometries a number of problems appear in the computation of the penalty forces. A big amount of primitives could lead in a computationally hard search of the exterior surface as illustrated in Figure 2.10 (a). Additionally, since the haptic point penetrates the object it is not always clear towards which surface must the force be directed (Figure 2.10 (b)). Another

conflicting situation is that thin objects are traversed by the haptic tool easily (Figure 2.10 (c)). When the haptic point passes more than halfway through the object, the reaction force's direction flips towards the other side. These undesired reactions are known as the *pop-through* effect and must be solved in some manner (Ruspini et al., 1997).

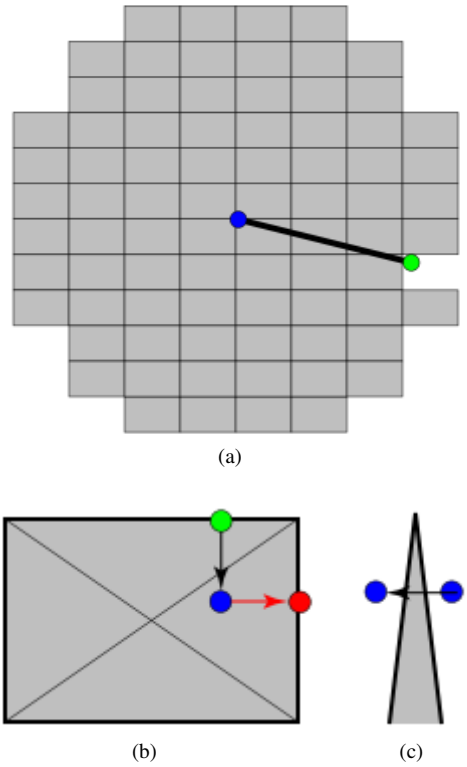


Figure 2.10: Drawbacks of penalty methods (Ruspini et al., 1997): (a) a global search of all the primitives may be required to find the nearest exterior surface, (b) the force could be directed towards the wrong surface, (c) small or thin objects may have insufficient internal volume to generate the constraint forces required to prevent the haptic tool from traversing the model

Penalty methods differ depending on the collision detection method used to identify the colliding primitives. In such a way, polygonal penalty methods are applied to scenes where the objects are represented by polygons (e.g. triangles or tetrahedra) since the collision detection process returns a set of intersecting

polygons as a base to compute the response. On the other hand, volumetric collision detection algorithms offer voxel-based information so the penalty forces to be applied are derived from voxels. Volumetric penalty methods solve these matters (McNeely et al., 1999)(Renz et al., 2001).

In order to address the problems of the penalty methods, constraint-based approaches offer the possibility of uncoupling the virtual representation of the haptic interface point and restrict its movements to the surface of the object. This method was first proposed by (Zilles and Salisbury, 1995a) and is known as the *god-object* method. This algorithm has been used to model interactions between a point-size god-object and complex polygonal models.

(Ruspini et al., 1997) extend the idea of the god-object location method in order to model extra features as force shading, friction, surface stiffness and texture by only changing the position of the uncoupled virtual point.

2.3 Collisions in Surgery Simulators

Section 1.1 has presented the different disciplines a surgery simulator covers. Amongst them, collision handling and haptics. The main objective of our work is focused on these stages, which apart from detecting interferences between the scene objects, give an accurate and realistic response to the colliding objects and the user. Our final simulator contains rigid and deformable virtual bodies (a skull and a brain tissue model). For that reason, the collision detection must be performed for both rigid and deformable models, and as the surgery tool is governed by a haptic device, the response involves a force feedback via the haptic device and a visual feedback between the virtual tool and the skull or brain.

Many haptic rendering and collision handling approaches have been proposed in different surgery simulators. Combining data structures and techniques as the ones explained at the beginning of this chapter, different algorithms can be constructed to give plausible answers to object interferences in simulators. Depending on many factors as the input data structures, output devices or the application's purposes the collision detection and response methods vary. Hereafter some examples are displayed.

2.3.1 Collisions in Craniotomy and Bone Drilling Simulators

Different simulation platforms have been developed which include rigid objects derived from patient data. As said before, the most used medical information format for craniotomy and bone drilling simulators is the CT scan. The CT is a diagnostic procedure that obtains cross-sectional pictures of the body. These pictures give material opacity units, which are displayed in the computer as voxel structures with density information. When manipulating such structures in order to find and solve interferences in the simulators, different techniques have been used. Some of them will be presented in this section.

B. Pflesser et all. (Pflesser et al., 2002) developed a system for virtual petrous bone surgery that simulates specific laterobasal surgical approaches. Anatomical objects are represented in a 3D rectilinear grid of volume elements derived from CT data. Different bone parts are defined by means of a semi-automatic segmentation approach. Cutting regions are modelled in independent data volumes using voxelization techniques and the voxelization is adapted to successive cutting operations. Additionally, the geometric description of the cutting tool is also converted into a volumetric representation. They combine the mentioned independent multi-volume representation with a visualization technique which reduces rendering artifacts. The haptic device used is a Sensable 3 DoF Phantom device.

From the same group was created the temporal bone surgery simulator TempoSurg (Petersik et al., 2002). This is an approach for haptic volume interaction where the anatomic models are also based on attributed voxels. The tool is represented by a number of sample points distributed over its surface. Concerning to the skull model, they construct its surface by a ray-casting algorithm based on the segmentation data. Each of the sample points of the tool is checked whether it collides with the objects or not. Additionally every point has an associated normal vector which is pointing to the inside of the tool. All colliding surface points are traced along the corresponding inward pointing normal until the surface of the object is found or the end of the normal is reached (Figure 2.11). Whenever a collision between the tool and static objects occurs, the direction and magnitude of the collision force are calculated. They are adapted in order to overcome frequency differences, and a vibration is also modulated onto the drilling force to improve the sensation of drilling.

The prototype of freehand-controlled bone drilling simulator presented by He and Chen (He and Chen, 2006) offers a realistic haptic rendering and an

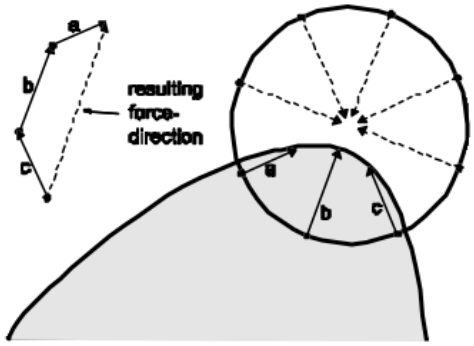


Figure 2.11: Calculation of collision forces on the surface: colliding points are traced along their normal until they reach the surface (Petersik et al., 2002)

efficient graphic rendering. This is achieved by using a hybrid model of volumetric and polygonal parts. The haptic rendering is based on volumetric data with high resolution and multi-point collision detection. The drill is represented as the union of a sphere and a cylinder. The collision detection is performed based on an OBB-tree structure of the volumetric model. Collisions between the burr and a voxel are detected by checking whether the distance between them is smaller than the burr radius. The collision between the cylinder representing the drill shaft and bone is simply detected by checking the distance between the drill shaft axis and a voxel. The force modelling used to simulate bone dissection introduces elastic and frictional forces with spring-damping.

Agus et all. (Agus et al., 2003) discuss a haptic and visual simulation of bone-cutting burr developed as a component of a training system for temporal bone surgery. They use patient-specific volumetric data derived from CT images with a resolution of $256x256x219$. However, the region where the operation takes place is represented by a volume of $256x256x128$ cubical voxels. They approximate the voxels representing the skull with spheres of the same volume to accelerate collision detection (see Figure 2.13), and the tool's representation is converted into a grid around the tip of $5x5x5$ for force computations. The cutting process is divided into two successive steps: the first one estimates the bone material deformation and the resulting elastic forces, given the relative position of the burr with respect to the bone. The second step estimates the local rate of cutting by using an energy balance between the mechanical work performed by the burr motor and the energy needed to cut the bone. Additionally, the force

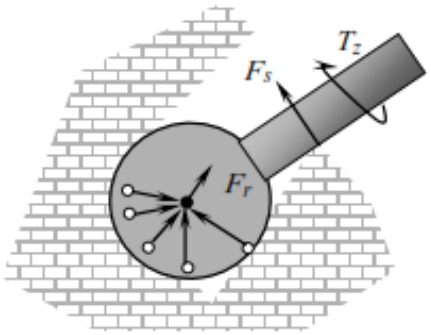


Figure 2.12: Representation of the drill for collision detection by He and Chen (He and Chen, 2006)

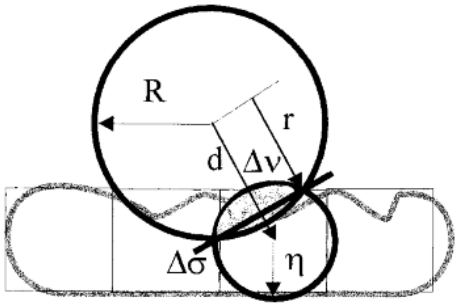


Figure 2.13: Voxels are approximated with spheres to accelerate collision detection (Agus et al., 2003)

restitution model incorporates friction in addition to elasticity.

Section 1.2 has presented the Intracranial Hematoma Craniotomy Simulator (Acosta and Liu, 2007), (Acosta et al., 2007) developed in the SimCenter. They describe real-time volumetric haptic and visual algorithms for a craniotomy surgical simulator. In this application the burr is represented by a point cloud that approximates its surface. It also has erosion points that determine the cutting capacity of each tool zone. The bone zones are represented with a voxel structure which encodes densities, density gradients and colours. Collisions are detected when the haptic points intersect voxels with a non-zero density value. Surface voxels are identified by following the vector associated to colliding haptic points

at voxel-sized intervals until a voxel with a non-zero gradient is found. A *tangent plane* is constructed for each surface voxel, which depending on the number of facets exposed to the surface give a normal vector and a penetration depth (see Figure 2.14). Those parameters are used to compute forces based on Hooke's Law. The algorithm's performance is based on the number of points used to construct the pointshell.

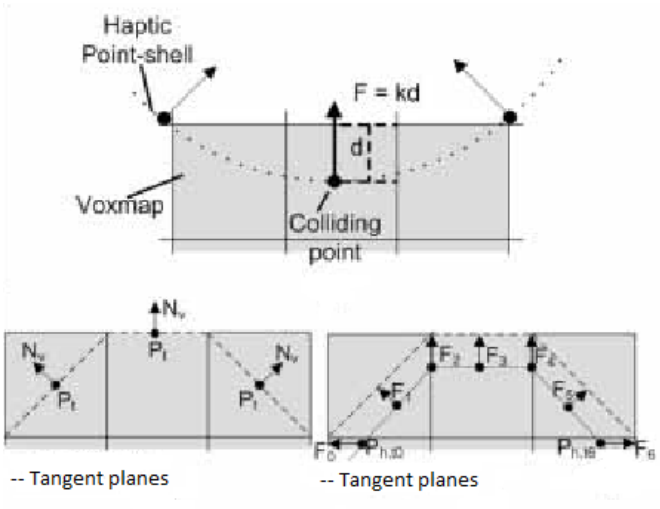


Figure 2.14: Pointshell interaction with a voxmap, generation of tangent planes and the force calculation in the Intracranial Hematoma Craniotomy Simulator (Acosta et al., 2007)

Morris et al. (Morris et al., 2006), members of the Stanford Biorobotics Lab, performed a framework for temporal bone dissection, using a hybrid data representation: bone is represented by volumetric data for haptic simulation of bone removal, and triangulated surfaces are used for graphic rendering. The drill is represented as a cloud of sample points, distributed uniformly around the spherical surface. Sample points are tested for intersection with the bone, and a ray is traced from each intersecting point toward the tool centre until the bone surface is found. Thus, the contact force to move that sample point out of the bone can be generated. Furthermore, when bone-voxels are removed as a consequence of the collision, the burred zone is re-tessellated by creating triangles that contain the centres of the new surface voxels as vertices. To give a more realistic haptic sensation, the vibration and sound of the instrument are added.

Other simulation environments transform image data into other specific

formats in order to manipulate them easier. Morris et al. (Morris et al., 2005) also proposed a simulator focused on craniofacial procedures. They transform data obtained from CT or MR (see Figure 2.15) into isosurfaces by using the Marching Cubes method (Lorensen and Cline, 1987). Isosurfaces are then capped using the 3D Studio Max software package, and a set of texture coordinates is also generated on the isosurface mesh. Afterwards, a flood-filling technique is used to build a voxel grid. Texture coordinates and surface normals are assigned to boundary voxels, and they are exported along with density information. This is used for haptic rendering. Finally, this voxel array is retessellated into a new surface used for graphic rendering. All this preprocessing time can be around fifteen or twenty minutes long.

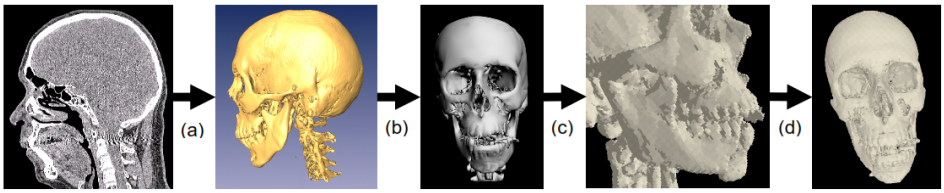


Figure 2.15: Data transformation process. Starting from CT data, they are (a) isosurfaced in a first stage, (b) capped and smoothed in a second phase, (c) flood-filled to generate a voxel array and finally (d) re-tessellated into a surface mesh (Morris et al., 2005)

Asenjo (Asenjo, 2008) suggests two different approaches to detect and give an answer to collisions for a bone milling and drilling haptic surgery simulator. The first one is called *The Line Approximation Algorithm*. In this method, the tool is represented as the combination of a sphere and a cylinder. The burr is approximated with a vertex in the centre of the sphere and the cylinder with an arbitrary amount of vertices along the centreline as seen in Figure 2.16 (a). The collision detection is based on checking density values of the points along the centreline of the tools tip. The intersection point is found by making a study of the tips trajectory from time step $t-1$ to t. When a collision occurs, a proxy point is determined based on a spring-model algorithm (Magnus, 2006). A vector which represents the spring's length and direction is also derived from the proxy and probe position. This information is used to calculate the haptic force at the proxy point. The torque at that point is also needed, which is derived from the intersection point on the shaft of the probe and the previous calculated intersection point. The second algorithm he proposes is called *The Surface Approximation Algorithm*. The fundamental idea of this algorithm is to put vertices on the

surface of the tool, both on the sphere and the cylinder. These vertices are used for collision detection and force and torque calculations. Collision detection is performed comparing the density values between the tool and the object. However, the new representation of the tool leads in a more stable haptic feedback since the force is computed as the sum of the forces of all the colliding vertices.

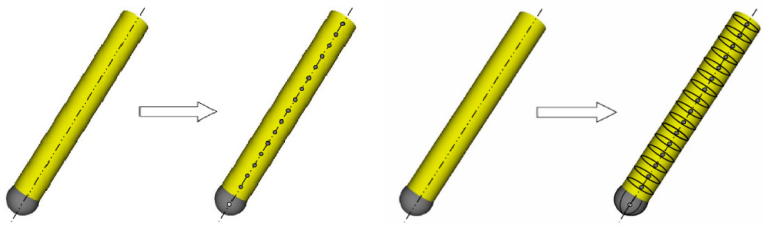


Figure 2.16: Discretization of the milling tool in (a) The Line Approximation Algorithm and (b) The Surface Approximation Algorithm (Asenjo, 2008)

2.3.2 Collisions in Neurosurgery and Other Deformable Simulators

Concerning to the collision techniques commonly used in neurosurgery simulators, a variety of methods have also been proposed in the last years. The data structures used to represent deformable brain models are usually tetrahedral meshes. Some simulators adapt these structures in order to add specific features as cuts. Let's review some of the techniques found in the bibliography.

Bielser and Gross (Biesel and Gross, 2000) presented a framework for the interactive simulation of surgical cuts. They base on tetrahedral volume meshes to provide topological flexibility. Collisions between the surgical tool and the tissue are detected by using AABB hierarchies adapted for deformable objects. The detection basically consists of two tasks: surface collision detection and volume collision detection. The first one is made by constructing an AABB hierarchy over the surface triangles, following the deformation of the object. The tree is locally updated by bottom up traversal starting from each deformed surface element. The size of each bounding box has also a small tolerance to skip updating for small positional changes of a child element (see Figure 2.17 (a)).

The intersection of the surgical tool and the tissue surface is computed by performing bounding box tests hierarchically by top-down traversal of the tree. After detecting the entry points, the volume collision detection starts by searching

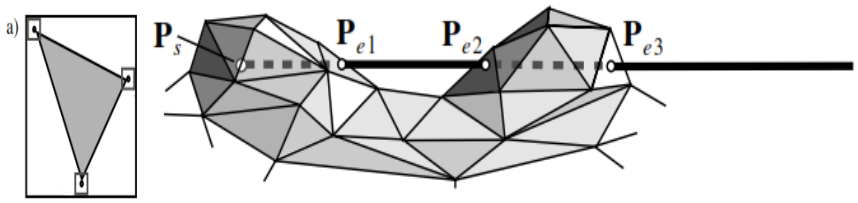


Figure 2.17: (a) Tolerances added to each node within the AABB, (b) The scalpel tip intersecting the tissue having a segment out (Biesel and Gross, 2000)

all active tetrahedra. Afterwards, representing the scalpel by a thin line, the trajectory of the scalpel tip is traversed. It computes all face intersections of tetrahedra lying between the current tip tetrahedron and the previous active tip tetrahedron. In a second step all the tetrahedra intersected by the swept surface are traversed in order to get the edge intersections. For haptic rendering and feedback, they devised a mechanical scalpel model which accounts for the most important interaction forces between scalpel and tissue.

An approach proposed by Brown et al. (Brown et al., 2001) is another example of a surgery simulator using a BVH for its deformable collision detection. It is a microsurgical training application which allows vessel suturing. Objects are represented by nodes connected by links. These are grouped into triangles in the surface but unrestricted below it. Collisions are performed by using distance computations based on a BVH of spheres. Edge-to-edge, edge-to-face and face-to-face collisions are achieved depending on the interacting objects and the task wanted to be executed.

A similar technique was later used for an interactive suturing and knotting environment with haptic feedback (Payandeh and Shi, 2010), proposed to be part of a gaming simulator for training surgeons. The BVH of the suture is constructed with bottom-up at successive levels of detail (see Figure 2.18) and different collision techniques are used for different object types: collisions between the suture and the grasper are determined by modelling the graspers as line segments and checking them against the suture's BVH. Needle and soft tissue interferences are found by defining a bounding box around the needle and analysing the mesh nodes inside it. Self-collision are achieved by a top-down search of two copies of the suture's BVH.

Other simulators convert object data into simpler hybrid structures in order to

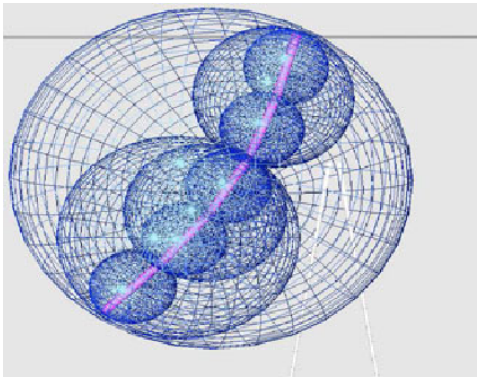


Figure 2.18: Bounding Sphere Hierarchy of the suture at successive levels, from (Payandeh and Shi, 2010)

handle collisions in an easier way. This is the case of the laparoscopic rectum surgery simulator developed by Pan et al. (Pan et al., 2011). Different tissues are classified into two types: the behaviour of membranes and fat tissues is simulated by a mass-spring model, while bowels are modelled as one-dimensional geometrical entities. The rectum is represented by a string of spheres, so collision detection is easily performed by using these spheres as bounding boxes.

Another option to determine collisions in a surgery simulator relies on the graphics hardware. Lombardo et al. (Lombardo et al., 1999) propose a real-time collision handling method for a surgery simulator aimed at training surgeons at minimally invasive techniques as laparoscopy. They represent the laparoscopic surgery tools as cylinders of constant section and varying length. What they basically do is a similar process to the one done in a scene visualization process: a viewing volume is specified (in this case it corresponds to the cylinders covering the tool between two consecutive steps) and the hardware is used to clip all the scene polygons according to that ”camera”. In such a way, only the intersection between the scene objects and the viewing volume is rendered. If nothing is visible, collision can be neglected. Otherwise the part of the object that the tool intersects will be shown. The problems this method carries are solved by using some features of the OpenGL graphics library.

Many other techniques have been proposed in order to achieve deformable collisions in surgery simulators. They base on the combination of basic collision detection techniques, depending on the data structures or the simulator requirements.

2.4 Discussion

This chapter has presented the basics of collision detection and response techniques and how different surgery simulators combine them to solve their collision handling problem.

As shown, many collision handling approaches have been proposed in different surgery simulators. Depending on different aspects such as the input models, their mechanical properties or the specific procedures wanted to be performed the collision handling methods vary. Focusing our attention on craniotomy and neurosurgery simulators, they frequently obtain their input data from CT or MRI images. CT data contain material opacity information and are displayed in the computer as voxel structures with density information. Commonly, surgery simulators convert these voxels into surfaces, hybrid models of polygonal and volumetric parts, generate additional structures that approximate the original structure and so forth. This is made in order to avoid the discontinuities that the proper voxelized structure has. However, knowing that the proper voxel structure derived from the CT is already a discrete approximation of the real data, additional structures or adaptations that approximate the already approximated structure can lead in an accumulation of possible numeric errors that can result trivial when working in the field of surgery. Intending to fill this gap, we propose a collision handling method that only uses the voxel information to give an stable and accurate response to the interfering objects.

Existing surgery simulators are usually focused on specific procedures. Many rigid and deformable collision handling methods have been proposed but, the combination of both matters is not commonly seen. Our purpose is to develop an accurate neurosurgery simulator that apart from offering an efficient collision handling method for voxel-based data, also leads with deformable bodies and combines both tasks in the same environment. Both collision handling techniques should also offer a realistic and stable haptic response.

Part II

Proposal

Chapter 3

Rigid Volumetric Collision Handling

Medicine, the only profession that labours
incessantly to destroy the reason for its existence
JAMES BRYCE

3.1 Introduction

This chapter presents our approach on rigid volumetric collision handling. This will be integrated into a Craniotomy Simulator which will be detailed in Chapter 6. Our main goal is to obtain a convincing haptic interaction with a rigid volumetric skull. For that purpose, an adequate collision detection algorithm is required, together with an accurate collision response and haptic control. The collision detection algorithm must efficiently detect the interferences in the environment and offer accurate information about collisions. This information is later interpreted by the collision response method and converted into a convincing force restitution. This chapter first describes the proposed rigid volumetric collision detection algorithm, to later present the implemented collision response and haptic control process. Finally, performed experimental results will show the performance of the methods in terms of time consumption and stability.

As previously mentioned, the Craniotomy Simulator requires collision handling for rigid objects. There are many methods and algorithms to solve the problem for rigid objects represented by all types of geometry as it has been presented in the previous chapter. This Craniotomy Simulator uses data

represented just by volumetric information obtained from real CT images without the need of converting them to triangle meshes, tetrahedral meshes or any other adapted structure.

Nowadays, most of the bone-drilling simulators convert volumetric data to specific adapted structures (McNeely et al., 1999) (He and Chen, 2006) in order to decrease force discontinuities. In our case, the visual module does not use any other information but the one about bone densities obtained from the CT. Our method gives a realistic haptic rendering by solving adversities instead of constructing a new hybrid or adapted geometry.

3.2 Collision Detection

The developed collision detection finds the interferences between a dynamic virtual tool and a rigid volumetric skull model. The skull bone is represented by a set of voxels with an associated density value. This value informs about the density of the piece of bone contained in each voxel. In such a way, voxels with a null density value correspond to empty zones in the space, while the ones with a non-empty value correspond to different bone areas with different densities. This has clear benefits in the case of requiring a distinction of different skull bone areas.

The role of the dynamic object is assumed by a simple sphere of the same dimensions as the milling tool. Its surface is surrounded by a cloud of points. From this point forward, we will refer to this point cloud as the "*pointshell*". Points on a sphere can be evenly or irregularly distributed. There are several techniques to generate a uniform discretization such as the Uniform Random Distribution, the Normal-Deviate Method, the Hypercube Rejection Method or the Trig Method. As mentioned before, points can also be distributed randomly by, for instance, using Particle Systems. We have used the Uniform Random Distribution by applying the equation in 3.1.

$$\begin{aligned} x &= r\cos\alpha\sin\beta \\ y &= r\sin\alpha\sin\beta \\ z &= r\cos\beta \end{aligned} \tag{3.1}$$

with α and β values inside the ranges

$$0 \leq \alpha < 2\pi, \qquad 0 \leq \beta < \pi$$

The chosen step between consecutive α and β values determines the number of points that finally constitutes the surface of the sphere. These generated points have an associated normal to the centre of the sphere, which will be used for the determination of the contact normal in the collision response process.

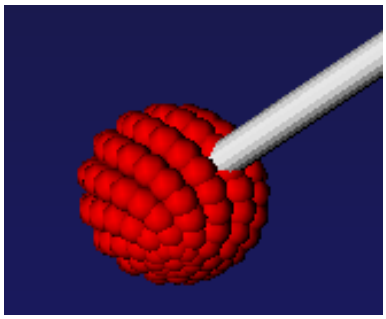


Figure 3.1: Pointshell surrounding the tool

As a preprocess stage, whenever the collision detection wants to be performed, data structures need to be initialized: to start with, the haptic module gets the current position and rotation of the haptic device. This information is assigned to the virtual model of the tool. From now on, this will be referred to as the "*proxy*" and it is covered by the previously mentioned pointshell.

Once the virtual tool is correctly located, the collision detection process starts its work. The general work flow of this module is shown in Figure 3.2.

As a first step, the collision module receives the movement made by user with the haptic device. At this point, even if directly applying the translation and rotation to the virtual tool could seem sufficient, it is not enough: some virtual object could intersect the whole action, so the proxy should not imitate that entire movement. In those cases, the virtual sphere only moves inside its possibilities. This is made by a process we have called *"sweep"*.

The concept of the *sweep* process consists in covering all the voxels from one point in the space to another, to determine whether the path crosses a bone zone or not. The route is divided into small pieces of the size of the smallest voxel side. Therefore, it is assured that no voxel will be skipped. The route is then covered piece by piece, comparing the density of the voxel in which it lies with a predefined isovalue[1]. Voxels with larger density than the isovalue represent bone

[1] Indicates the density value level of the voxels attenuation values done by the graphic rendering of the 3D object

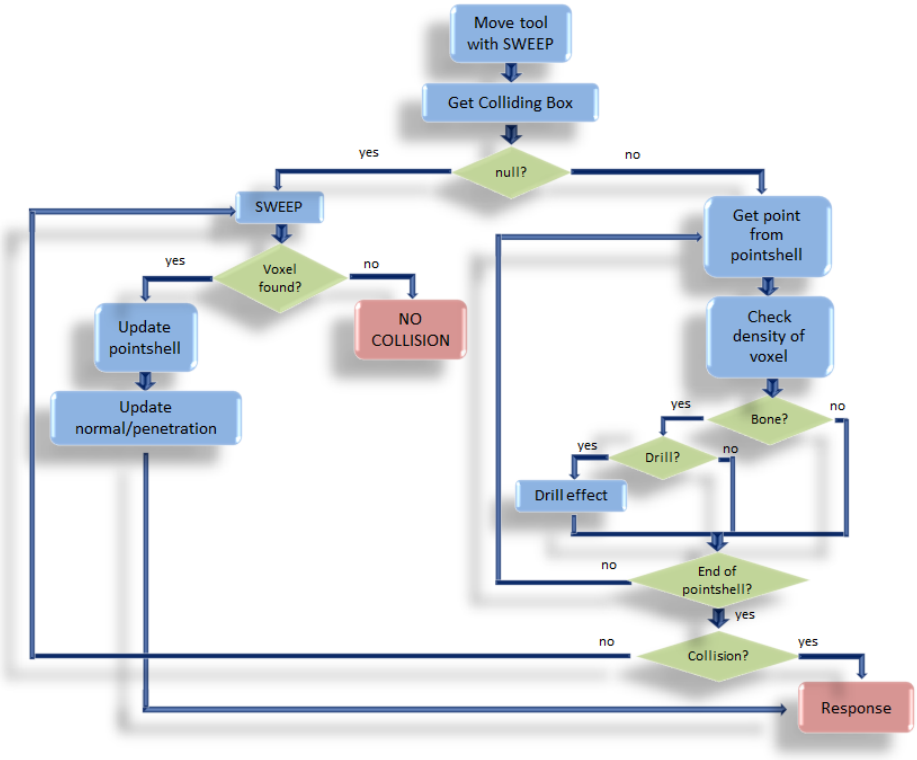


Figure 3.2: Collision detection control flow

zones. Thus, if a bone-voxel is found before arriving to the objective position, it means that the path collides with an object so the tool must stop its trajectory at that point as shown in Figure 3.3. It is located in such a way that it does not penetrate the bone zone completely but it lightly touches it. Its exact location depends on the radius of the sphere.

In this way, the first step of moving the virtual tool is performed by means of a sweep of voxels which starts from the previous position of the virtual tool in the direction of the new position. It stops when a full voxel is found or when the whole path is covered, and the new proxy position is set at that point (see Figure 3.4). This can result in a ”gap” between the locations of the real tool and the proxy which is assumed and controlled. The real tool position coincides with that one measured in the device, while the ideal proxy position is consistent with the restrictions of the scene. The act of uncoupling the positions of the real haptic tool and the proxy is called *Virtual Coupling* and is an extended haptic rendering

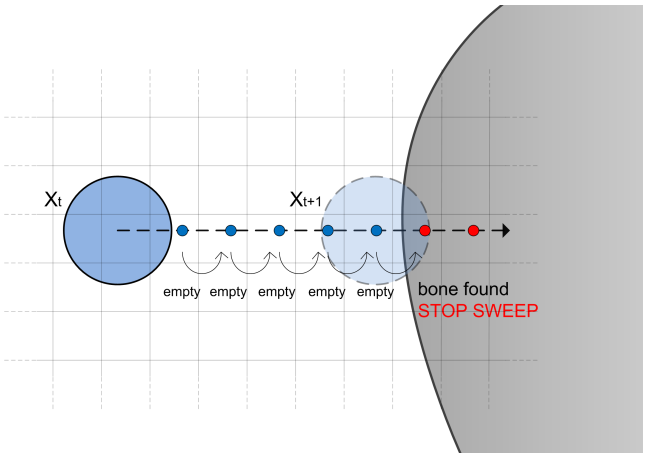


Figure 3.3: Sweep process

technique (Colgate et al., 1995). The nature of this gap will be important when deciding the course of the algorithm in some occasions. This will be detailed later.

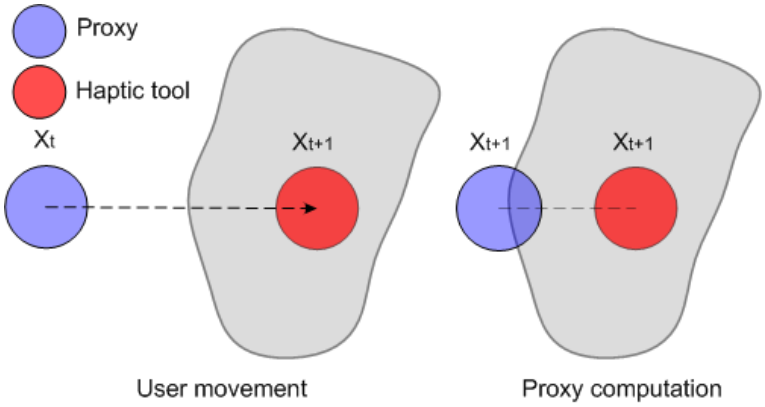


Figure 3.4: Sweep with the haptic and proxy before moving

Once internal structures have been updated with the current state of the scene objects, the sphere and pointshell involving the tool are updated and a *colliding box* is defined. We refer to the *colliding box* as the cuboidal zone where the AABB of both the skull and the tool-sphere interfere (see Figure 3.5).

This first phase of delimiting the whole problem to a so-called colliding box accelerates the process: if the box is null, the collision process does not continue checking for interference between the objects. Thus, it avoids checking

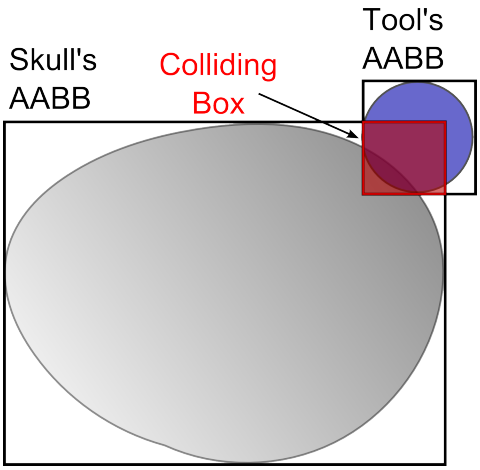


Figure 3.5: The colliding box is defined as the intersection of the skull's AABB with the tool's AABB

for interference when the models are far away from each other.

At this point, and if the colliding box is not empty, the cells inside it are checked for real intersection. All the points belonging to the pointshell are checked in order to take the ones lying inside the colliding box. For every point lying inside the box, the first question is whether it belongs to a bone-voxel or not. This is checked by finding the voxel in which it lies and comparing its density with the predefined isovalue. If the cell represents a bone zone, it means that the cell belongs to the skull and is also located inside the tool-sphere. In conclusion, the tool is touching the skull. Thus, the mentioned voxel and point are saved, which will be later used to calculate the contact normal and penetration depth.

If a collision is found and the drilling flag is active, the removal of the skull bone is performed here by decrementing the density associated to the colliding voxels by 1.0. In our case, the isovalues of the skull bone vary between 300 and 500.

However, this method has some problems. Working with an space lag between the real position of the tool and the proxy carries undesirable situations in some circumstances, which have been faced by the use of the sweep process. The nature of the problem is basically one, but it can happen in two different situations:

- The AABBs of the skull and the tool do not overlap: If the colliding box is empty, no collision is happening. However, this is not always true. Let's see

a circumstance in which having an empty colliding box the user must still feel a force: when the real and proxy positions of the tool differ. This can happen when a collision is happening in the previous frame, and the proxy position has been moved to the surface of the skull while the real position of the tool is inside it. The movement made by the user from the previous frame to the current one can position the proxy in a non-colliding place. However, the real tool can still be colliding. For that reason, the user must keep feeling a force, and the proxy must keep touching the skull (see Figure 3.6).

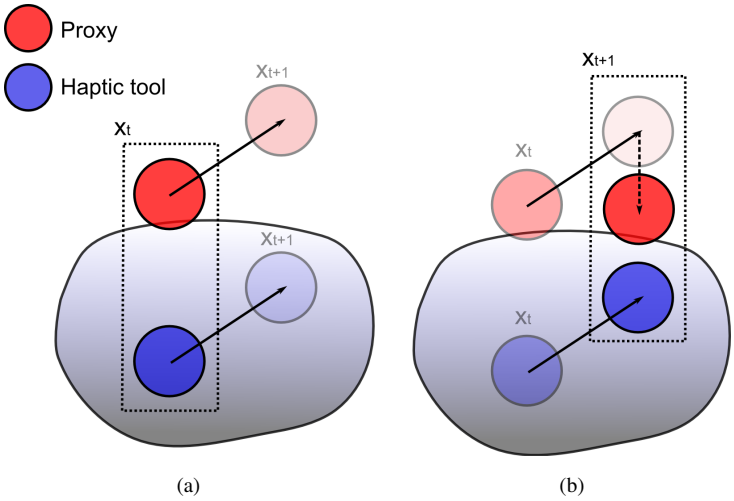


Figure 3.6: The real (blue) and proxy (red) positions differ. Once the movement is done, the proxy is free of interferences but the real tool is still inside the skull (a). The user must keep feeling a force, so a sweep process is needed (b)

- The AABBs overlap, but no colliding point is found: This is basically the same problem as the previous one. Even if in this case the proxy lies inside the skull's AABB, no colliding point is found. So it does not really collide with the skull. The proxy could have been moved to a non-colliding position while the real tool is still in collision. The user must also keep feeling a force and the proxy must be placed in the surface.

An additional sweep stage solves these matters. A sweep process from the proxy position to the real tool position is thrown (Figure 3.6). The reason for doing this is that the force wanted to be felt is the one related to the real movement

of the user's hand, so even if the visual representation of the tool (the proxy) is placed in a different location, it always tries to match the real one. In such a way, if the sweep does not find any skull voxel along its path, there is absolutely no interference between both tools. The real tool is not in collision and the proxy can be moved along the path to the real haptic position. Both tools can be matched, and the collision can be discarded. On the contrary, if some skull voxel is found, the user must keep feeling a force, and the proxy cannot be moved to the real position as something is placed between them. The proxy would visually traverse the skull. In that case, the contact normal is considered the inverse of the sweep direction. Since the proxy is relocated to the position where the sweep has stopped, this is done with a predefined small penetration depth.

The sweep can be adapted to the scene, objects and circumstances. If it is a-priori ensured that the skull does not contain cavities of smaller dimensions than the tool's diameter, and if the radius is relatively small, the sweep can be done from the centre of the tool to the new predicted centre, just following a line. Otherwise, as seen in Figure 3.7, the sweep should be made considering the path as a cylinder, not as just a line. It could also be done by defining the path as a string of spheres with the same radius as the tool and searching for interferences along the path. Obviously, this would carry a higher computational cost.

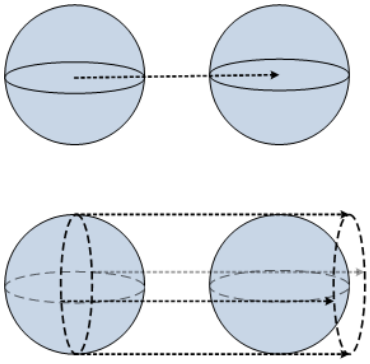


Figure 3.7: The sweep can be made with a cylinder instead of a simple line. This is not necessary in environments where the radius of the tool is relatively small

Information about the colliding voxels and points and the new proxy position if the sweep has moved the tool is sent to the collision response module in the next stage. A haptic and visual response are needed, so all this information has to be processed and used to give a realistic feedback in both senses.

3.2.1 Visual feedback

The haptic interface does not have infinite stiffness. For that reason, we cannot stop the haptic interface point from penetrating the virtual objects. It is proven that the visual sensory channel can strongly distract the user's perception of feeling a force. Thus, if the virtual tool visually remains in the surface of the deformable body even if the force limitations are exceeded, the perception of stiffness is stronger. This is what we call *visual feedback*. The idea is to define two tool objects: one matches the movements of the haptic device but is not visualized, and the other is located where the tool should be if the skull were completely impenetrable. The latter is what we have been calling *proxy*.

In the bibliography, a method called *"god-object"* (Zilles and Salisbury, 1995a) modifies the location of the proxy tool and forces it to follow the laws of physics by staying where it would be if the haptic interface and the virtual object were infinitely stiff. This method is what we have adapted and used to give a realistic visual feedback to our algorithm. Our proxy object is the one which suffers the location changes determined by the god-object location method, while the real tool can freely move around the environment but is not visualized.

The visual feedback in our method is applied after the collision handling process and before giving a haptic response to the user. Once interferences are identified, the visual representation of the tool is updated, and its new location is determined depending on the results of the detection process. Let's analyse the different situations in which the collision detection process can end and how the visual feedback behaves for each one.

Having a look to Figure 3.2 which illustrates the collision detection's workflow, the detection process can reach two different states at the end of its flow: collision or no collision. In the case of no collision, no additional visual feedback is needed. The proxy position is just matched with the haptic position.

However, the state of "collision" can be reached from two paths: a sweep can have moved the proxy to a feasible position and thus the proxy tool is placed at the object's surface, or the collision detection process has followed the normal flow and no rectifying sweep has been performed. It is obvious that in the first case (the sweep process has moved the proxy position) the visual feedback only involves moving the proxy to the new location.

Nonetheless, if a collision has been detected and the virtual tool's position has not been rectified by a sweep, the performed visual feedback becomes a bit

more sophisticated. In that case, it is possible that the haptic tool is positioned inside the skull while the proxy is somewhere in the surface (where the moving sweep has placed it). If the haptic tool is in collision the proxy must be moved to its correct location in the surface. Thus, an adaptation of the previously cited god-object location method (Zilles and Salisbury, 1995a) is applied which, having a constraint plane, provides the virtual tool with a new projected position. Our algorithm does not contain surface information. So the constraint plane used for the god-object location method is derived from the contact normal calculated for the force restitution. Even if this method gives a feasible visual feedback, the collision detection method has been performed for the current proxy position, so the new location does not assure no interference will be happening with any other plane which is not in collision now. No collision with the constraint plane is ensured, but the proxy can interfere with another plane which has not been taken into account in the current stage.

For that reason, an additional rectifying step is again needed at this point. Actually, it is necessary every time the proxy position wants to be updated directly: when a god-object location is defined as well as when no collision is found and the proxy wants to match the haptic position. It consists of sweeping the voxels from the previous proxy position to the new one given by the god-object location method or by the haptic position. If the new position involves crossing a bone zone, the sweep stops and locates the proxy at that point.

3.3 Collision Response and Haptic Control

The advent of haptic devices into VR environments provides a more realistic sense of immersion to the user, apart from increasing the interactivity. The user can feel a physical stimulus when the scene objects collide. Additionally, different textures can be perceived for each object as in real world.

In the case of surgery simulators, the student or surgeon could perform the surgery tasks as skull drilling or interacting with the brain tissue without any other feedback but the visual one. However, the accuracy of the application increases significantly if the user is able to feel the force that avoids him to penetrate into other virtual objects. The accuracy required in surgery tasks makes the use of haptic devices more necessary.

The computation of the collision response is done using the information given by the collision detection module. This module constantly verifies collisions

between different objects of the scene and calculates the interfering primitives. The interfering primitives returned from the rigid and deformable collision detection modules differ, as the structures used to detect collisions are different in both cases. Anyway, the returned information is used by the collision response module to compute a set of forces that will act against the prohibited movement of the user.

This section analyses the rigid haptic collision response problem by a short description of the different parameters used to determine a solution for the interfering objects in the scene. This solution is obtained from the geometric information provided by the collision detection module. The given solution is based on the penalty forces method described in Section 2.2.

Figure 3.8 illustrates the control flow of the whole rigid volumetric collision handling process. As it is shown, the collision response module calculates the final force to be sent to the haptic device based on the data received from the collision detection module.

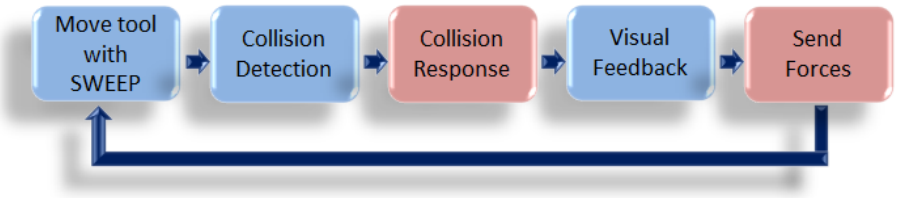


Figure 3.8: Control flow of the rigid volumetric collision handling process

As mentioned above, the information sent by the detection module is different for the rigid and deformable collision detection modules. In the rigid case, as the skull model is represented by voxels, no triangles or surfaces are known. For that reason, the rigid collision detection module returns the list of skull-bone voxels which are in collision. Apart from that, for each colliding voxel, the list of points belonging to the pointshell that are actually colliding with it are given.

The collision response module computes the force restitution through the haptic device. This is proportional to the penetration of the intersecting objects. The system calculates and returns a three-dimensional force. Due to the lack of torques in our system, the calculation of the force can simply be done with the force direction (contact normal), penetration depth and a stiffness constant (k). The simplest way to model the restitution force is by the use of a spring model

defined by the Hooke's law of elasticity:

$$\mathbf{F} = kp\mathbf{n} \tag{3.2}$$

where k is the stiffness constant, p is the penetration depth of the collision and $\mathbf{n}$ is the contact normal.

However, a viscous component is frequently added to the elastic model defined previously. This model is called *viscoelastic* and adds viscosity by adding velocity information to the deformable force restitution. This contributes to the system's stability and improves the contact feeling. The hereafter equation is applied:

$$\boldsymbol{F} = kp\mathbf{n} + b\mathbf{v} \tag{3.3}$$

where k and b are the stiffness and viscosity constants respectively, p is the penetration depth, $\mathbf{n}$ is the contact normal and $\mathbf{v}$ is the velocity of the haptic device.

The contact normal and penetration depth are calculated based on the voxel and point information received from the detection module. Additionally, the contact perception of the user is also affected by the chosen stiffness constant k. The larger this value is, the more rigid the contact will be felt. Thought, the system's stability cannot be assured for every chosen stiffness value. The velocity information is also required which can be obtained directly from the haptic device. Finally, viscosity constant b values must also be adjusted in order to maintain the global stability of the system. Let's now analyse how the contact normal and the penetration depth are obtained.

Logic says that the contact normal should be the direction correspondent to the surface normal touched by the virtual tool. However, the difficulty of this method is the lack of surface data in the skull model. The object is represented by density voxels, so there is no additional geometry information available. The main adversity comes when getting the surface normal. In our case, the penetration depth of each colliding point and the contact normal for each colliding voxel is computed based on (McNeely et al., 1999): the penetration is calculated as the distance d from the colliding point to a plane called *tangent plane* (see Figure 3.9). This is defined as the plane that passes through the centre point of the voxel and has the same normal as the colliding point's associated normal.

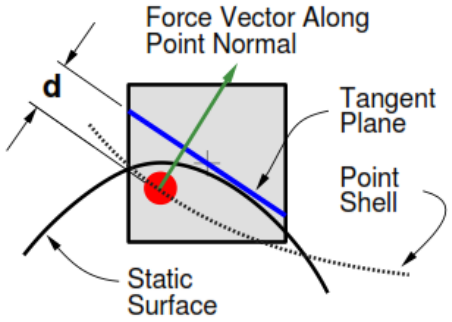


Figure 3.9: Penetration depth and contact normal calculation (McNeely et al., 1999)

Nonetheless, while they compute the final force as the sum of local forces obtained from multiple colliding voxels, we obtain the contact normal and penetration for each single colliding point and compute an unique normal and an unique penetration depth from all of them. Then, we use these values to determine a single force. This was first proposed by (Renz et al., 2001) in order to solve the problem illustrated in Figure 3.10. When the colliding point crosses the voxel border, the local penetration is reduced to zero so the force returned by the local collision is also zero. This influences the overall collision force. In such a way, based on the list of voxels and points received from the collision detection module, the resulting force is computed by using the average of the contact normals (Equation 3.4) and the maximum penetration.

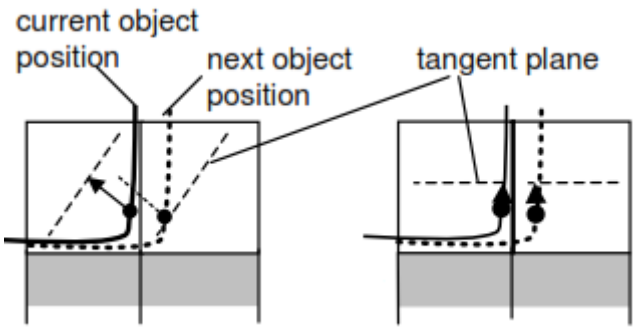


Figure 3.10: Adapted force calculation (Renz et al., 2001)

$$\mathbf{n}_{avg} = \frac{\sum_{i=1}^{m} \mathbf{n}_i}{|\sum_{i=1}^{m} \mathbf{n}_i|} \tag{3.4}$$

where $\mathbf{n}_{avg}$ is the average normal vector, m is the number of voxels in collision and n_i the contact normals of each colliding voxel.

Thus, the contact normal used to avoid the collision is defined as the average of the contact normals of all the colliding voxels. Regarding to the penetration depth, it is determined as the maximum of the average penetrations of the points colliding with each voxel. Namely, for each colliding voxel all the points colliding with it are searched and the average of their penetrations is taken. Then, the maximum of all the average penetrations is the one chosen to represent the penetration depth of the whole intersection. See Algorithm 1.

Algorithm 1 Rigid penetration depth

```
1:  function RIGID PENETRATION(voxels, points)
2:      dist ← 0
3:      penetration ← 0
4:      for i = 0 → voxels.size() do
5:          if points[i].size() > 0 then
6:              for j = 0 → points[i].size() do
7:                  dist ← dist + getDistance(points[i], voxel)
8:              end for
9:              dist ← dist/points[i].size()
10:             if penetration < dist then
11:                 penetration ← dist
12:             end if
13:         end if
14:     end for
15: end function
```

Even if the penetration depth is calculated in the collision response module, it could be necessary to modify it if a visual feedback is required. In those cases, the penetration depth is the distance between the centre of the haptic tool and the centre of the proxy. This will lead in larger force restitution the more the user penetrates the skull, even if the visual feedback does not let the user see it.

3.4 Experimental Results

In order to evaluate our rigid volumetric collision handling method, some experiments have been carried out. This section analyses the behaviour and validity of our algorithm, as well as the parameters that most affect its performance.

One of the biggest purposes of haptic systems is the avoidance of discontinuities in order to offer a pleasant experience to the user. The first adversity in this way comes with the proper models, since they are represented by discrete structures that implicitly add discontinuities.

Apart from that, the response time is also a crucial aspect to offer an accurate feeling. Haptic systems usually require a minimum frequency of $1kHz$ to offer a real-time interaction. In order to fulfil this necessity, the collision module's frequency should be as close as possible to the haptic module's frequency. This means that the collision module should finish its work in at worst $1ms$ time. Otherwise, the collision information sent by the haptic module to the haptic device would be out to date, so the forces felt by the user would not fit the real environment's situation.

With the aim of evaluating our methods in terms of continuity and stability, we have analysed two main aspects: the time consumption of the collision handling algorithms and the stability of the force parameters they return. The time consumed by the collision handling methods will show whether the required frequency can be achieved or not and which factors alter it.

Our first goal is to determine the frequency of the collision handling algorithms. We are also interested in the time consumption differences between input models with varying tessellation. When comparing the behaviour of an algorithm depending on the used models, an identical path is generally used. We call *path* to a sequence of discrete consecutive positions in the scene. In other words, a trajectory is defined by moving the virtual tool around the environment, so the same tool path is analysed for the same algorithm with different models. In such a way, the comparison is done by only varying the parameter we are owing to compare.

A path can be defined by different methods. We have used the proper haptic device to move the virtual tool around the environment and record the covered positions. Different input models could be tested differing the number of primitives, or the same object with different tessellations could be loaded. We have

opted to load the same structure varying the number of primitives (voxels) instead of loading different objects for a simple reason: when testing different models, some recorded locations that result in plausible colliding positions for one model could lie in unreal positions for other models. For instance, they could lie in some point inside the proper model. Or a no-collision situation could happen, which is not relevant for our study. However, this can also happen when loading the same object with varying number of primitives, if the surface of the model is complex. A different number of primitives also involves a bigger or lower precision in the surface of the model, which could lead in the problem we are trying to face. In order to avoid this problem, we have first done the experiments with cubical models which, obviously, have a plane surface. In this way, the recorded tool trajectory will cover plausible positions whichever the tessellation is.

The cubical models will help us to identify the possible problems of our algorithms. Afterwards, the time consumption of our methods with some skull models will be shown.

Regarding to the force parameters, the returned force parameters for consecutive frames will be shown in order to determine the stability of the system.

3.4.1 Time Consumption

A low computational cost of the rigid volumetric collision handling method is trivial to achieve the minimum frequency required by the haptic interface. Collision detection and response must be done in at most $1ms$. One of the factors that strongly affects this time is the size of the voxels composing the input rigid model. The voxel-size alters the collision times in the following aspects:

- Sweep process: small voxels prolong the sweep process since the step depends on the voxel-size. The smaller the voxel is, the smaller the step will be. Consequently the sweep needs more steps to cover the entire path, which makes the process longer. Anyway, the paths to cover are generally short since the surgeon is supposed to do small movements with light contacts. Therefore, times to perform this stage are generally small and in our particular case can be even rejected.

- Calculation of the contact normal: the contact normal is computed based on the colliding primitives returned by the collision detection module. Small voxels imply a larger amount of colliding voxels. In this way, a large amount of colliding voxels lengthens the computation of the normal.

- Calculation of the penetration depth: since the penetration depth also depends on the colliding voxels, this process would also be longer. However, the penetration depth's calculation also checks the points contained in each colliding voxel. As small voxels contain a small number of points inside them, the computation is shorter in this way.

With all this information, it seems that small voxels generally lengthen the collision handling process. Nonetheless, if the part of the penetration depth's calculation relative to the colliding points has a strong charge in the algorithm, the time consumption could be balanced. For that reason, the time consumed by models with a different voxelization is not clear. We have performed some experiments in order to see the computational cost differences between different rigid models varying their voxelization.

The number of points surrounding the virtual tool (number of points contained in the pointshell) also affects the time consumption. A large amount of pointshell points results in higher collision detection times. However, we are more interested in examining the tessellation of the input models rather than the tool's point density. This parameter can slightly affect the consumption time but, from a minimum number of points onward, the accuracy of the algorithm does not vary pretty much. As explained in the collision detection section, we have used a Uniform Random Distribution of points (see Equation 3.1 defined at the beginning of this chapter). The chosen step for α and β is 10.0 in the rigid case, which leads in 466 points surrounding the sphere. Thus, the mentioned step was the one that best time-accuracy balance offered. This value is maintained for all the rigid experiments.

Table 4.1 describes the cube models used to compare the algorithm's time consumption varying the voxel size. All the models have the same size, so varying the voxel-size means increasing or decreasing the number of primitives per axis. All the voxels are cubical so the voxel-size is only given for one axis. All this information is summarized in the table.

It must be noted that each experiment is performed by covering the same recorded tool path for all the examined models. Intending to be faithful to reality, this path is recorded with the proper haptic device in order to cover plausible positions.

The initial experiments are done with rigid volumetric cube models which help us analyse the behaviour of the algorithm. Afterwards, volumetric skull models will be examined in the same way in order to determine the voxelization

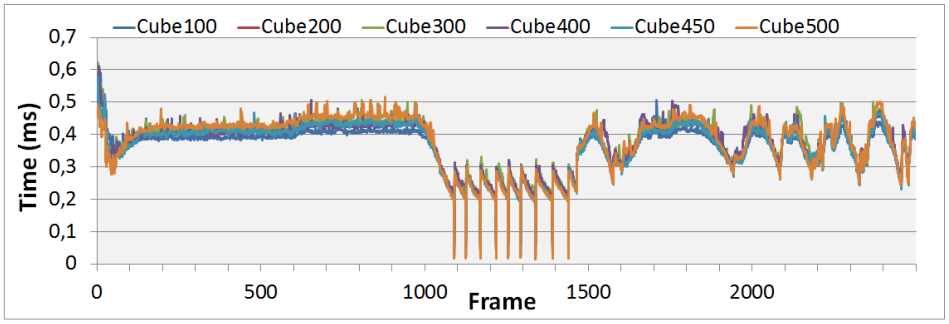
model	dimensions	voxel-size
Cube100	100x100x100	2.000
Cube200	200x200x200	1.000
Cube300	300x300x300	0.667
Cube400	400x400x400	0.500
Cube450	450x450x450	0.444
Cube500	500x500x500	0.400

Table 3.1: Description of the cube models, the number of voxels per axis and the consequent voxel-size

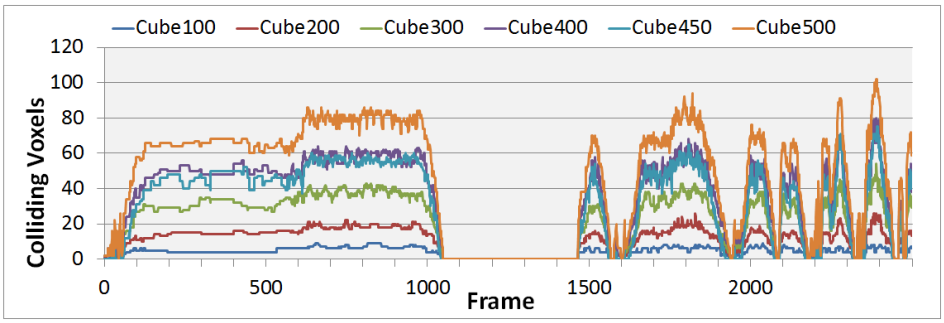
level of skull models our algorithm can support. Additionally, we have divided the experiments of both types of models into two different tasks: touch and drill. The first experiment is done by only touching the rigid model but not drilling, while the latter keeps the drilling flag active all the time. Figure 3.11 (a) shows the collision handling times in 2500 consecutive frames for the previously described cube models in the first case: touching the skull without drilling. The number of voxels in collision for each analysed position is given in Figure 3.11 (b).

As can be seen in the first graph, all the collision times lie inside the barrier of $1ms$. This means that the required minimum frequency is achieved for all the analysed cube models. Apart from that, even if the difference is minimal, it seems that smaller voxels return higher collision times. As expected, (b) confirms that small voxels imply a larger number of colliding primitives.

Frames $1048 - 1465$ seem to illustrate a range of non-colliding positions, since the number of voxels in collision is null all over the range. However, a collision is actually happening even if the number of voxels returned by the algorithm is null. The reason for this goes back to Chapter 3 where the sweep process has been introduced. As explained, our algorithm activates the sweep process in order to solve two conflicting cases in which the positions of the proxy and the haptic tool differ: in the first case the AABBs of the rigid model and the proxy do not overlap, but the collision cannot be discarded since the haptic tool could be colliding. For that reason, even though the AABBs do not overlap, the proxy and the tool cannot be directly matched and the sweep process places the proxy at its new plausible position. In the second conflicting case the AABBs of the rigid body and the proxy do overlap but they are not colliding. Once again, the proxy may not share any space with the object while the real haptic position is inside it. So the sweep relocates the proxy again. In both cases



(a) Collision handling times



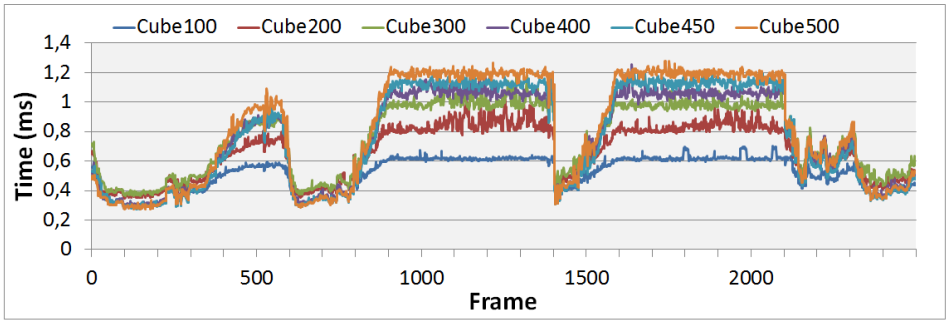
(b) Number of voxels in collision

Figure 3.11: Comparison of the collision times in a non-drilling task for cube models with varying tessellation

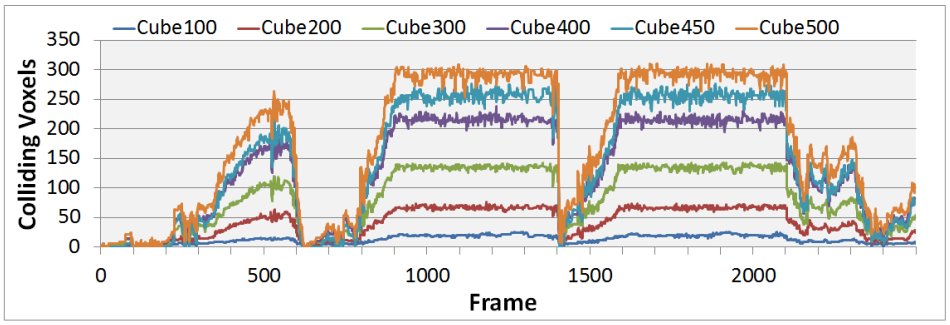
the force parameters related to the collision (the contact normal and penetration depth) are directly computed without the need of calculating the colliding voxels. This is why the number of voxels in collision is said to be null but it is truly not.

The times spent by the collision handling module in the mentioned range suffer some variations. The reason for this is that when the algorithm detects that the AABBs of both models do not overlap (first conflicting case), the only thing to be done is the sweep process and the direct computation of the force parameters. So the time spent by the algorithm to solve such situation is really low. The second conflicting case does not discard collisions until all the points of the pointshell are checked, so the time needed to finish it is slightly higher. So the almost null times shown in the graph correspond to the first conflicting case while the rest of the values inside the $1048 - 1465$ range illustrate the second one.

The collision handling times and colliding voxels during the drilling process are given in Figure 3.12. In the same way as with the non-drilling task, these graphs display the times and the number of colliding voxels for 2500 consecutive positions, but in this case the drilling flag is active during the whole experiment.



(a) Collision handling times



(b) Number of voxels in collision

Figure 3.12: Comparison of the collision times in a drilling task for cube models with varying tessellation

In the drilling task the time differences between different tessellations are more meaningful than in the non-drilling task. Drilling cube models with very small voxels consumes more that $1ms$ in some cases, while the same task with models composed by bigger voxels can be easily performed in the required time. Comparing Figures 3.11 and 3.12, it is clear that drilling is computationally more expensive than just touching. But, what does really happen in the drilling task to be more expensive than touching in terms of time consumption? At first sight, it might seem that the only difference is the activation of the drilling flag, which decreases the density of the colliding voxels until they disappear. Nonetheless,

there is another difference: the number of colliding primitives. When drilling bone the voxels in collision are removed so a hole is created inside the body. As a result, the drilling tool gradually enters the hole and it obviously gets in contact with a higher number of primitives (see Figure 3.13).

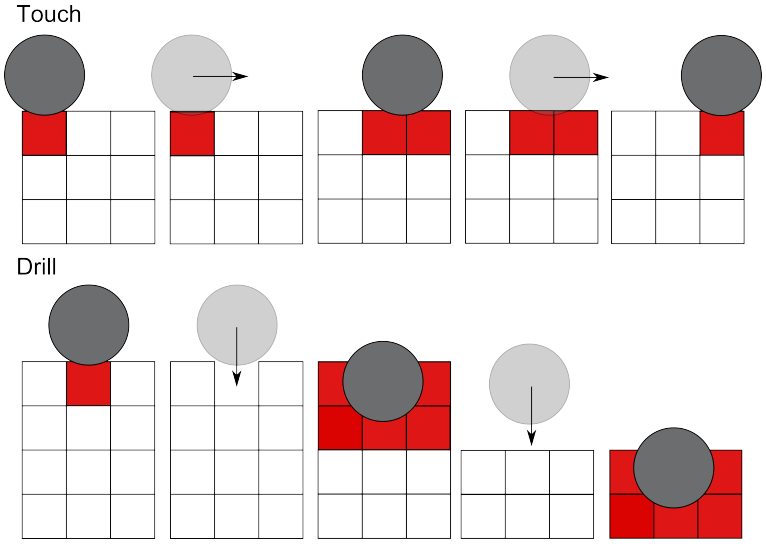


Figure 3.13: Colliding primitives in the touching and drilling process

The drilling process detects up to 300 voxels in collision in the Cube500 model while the non-drilling experiment does not pass beyond 100. So when the sweep process is not performed, the number of primitives needed to traverse in order to calculate the contact normal and penetration depth is bigger. This obviously leads in a heavier computational cost.

As mentioned before, when the drilling task is being accomplished the sweep process of the collision detection is rarely needed. It has also been stated that the sweep implies the direct calculation of the force parameters. So the sweep skips all the process of computing the contact normal and penetration based on the colliding voxels and points. Knowing that the time consumed by the sweep process is much lower than the computational cost of computing the force parameters, the activation of the sweep process shortens the collision handling time.

Since the rigid volumetric collision handling method has been designed to be integrated into a craniotomy simulator, the rigid input models are skull models derived from real CT data. Once the behaviour of the collision handling algorithm

has been analysed with simple cube models, we present some experiments carried out with skull models. What we have done is to resample a real skull model obtained from a CT into skull models with almost the same shape and size but different voxelization levels. In such a way, the same recorded tool path can be used to compare the effect of varying the voxelization without varying the covered positions. The original skull model contains $216x128x142$ voxels of size $0.984x1.584x0.997$. Table 3.2 describes the used skull models, their dimensions and the size of the voxels. Once again and in order to simplify the comparison, the chosen voxels are cubical.

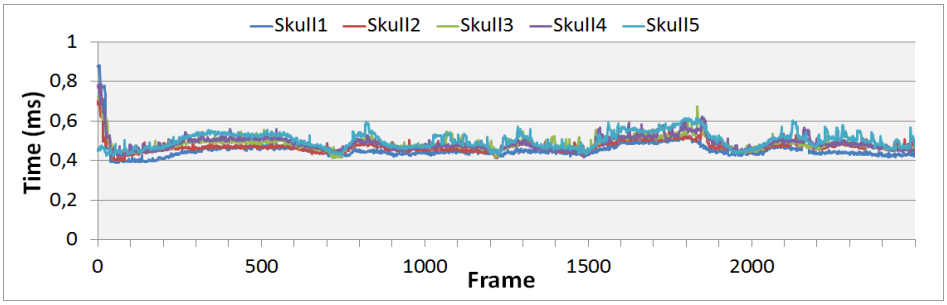
model	dimensions	voxel-size
Skull1	143x135x95	1.500
Skull2	171x162x114	1.250
Skull3	213x202x142	1.000
Skull4	284x269x189	0.750
Skull5	425x403x283	0.500

Table 3.2: Description of the skull models, the number of voxels per axis and the consequent voxel-size

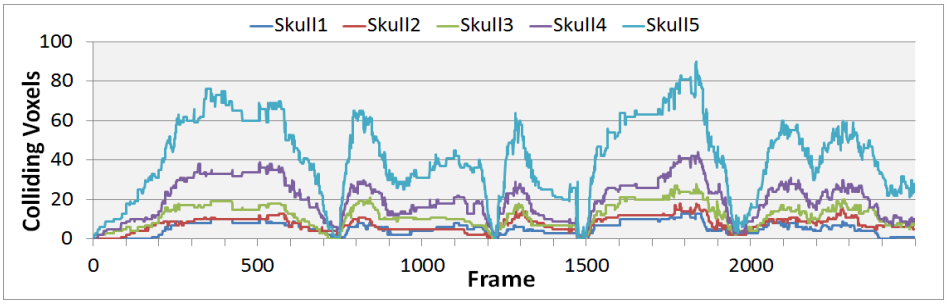
The system's accuracy strongly depends on the number of voxels contained in the skull model. The force parameters returned by the collision handling algorithm are closer to reality the more voxels the models contain. In contrast, a high number of voxels increases the time consumption of the collision handling method. Therefore, a compromise must be found. The idea is to increase the number of voxels as much as the collision handling algorithm is able to support computationally. The next graphs illustrate the rigid volumetric collision handling times obtained with different skull models. Figure 3.14 corresponds to a non-drilling task and the graph in (b) gives the number of voxels in collision for each frame.

The results show that even the Skull5 model which contains more than 48000000 voxels can be processed in less than $1ms$. As happened in the non-drilling task of the cube models, even if the number of colliding voxels differs for different models, the collision handling times are similar whichever the voxelization is.

Let's now see the results obtained in the drilling task with the same skull models. Seeing the results obtained with the cube models it can expected that the time consumption in this case will be larger than in the non-drilling task.



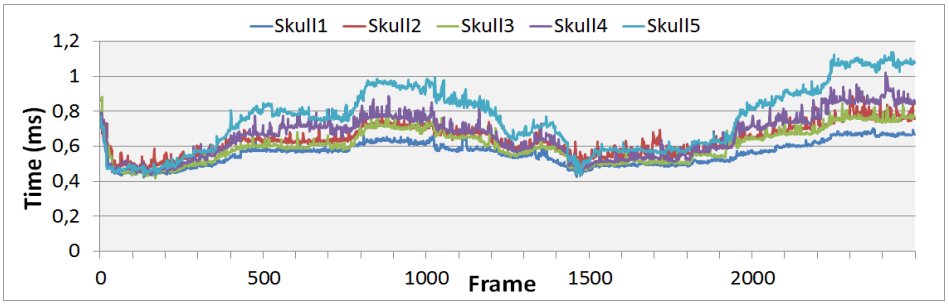
(a) Collision handling times



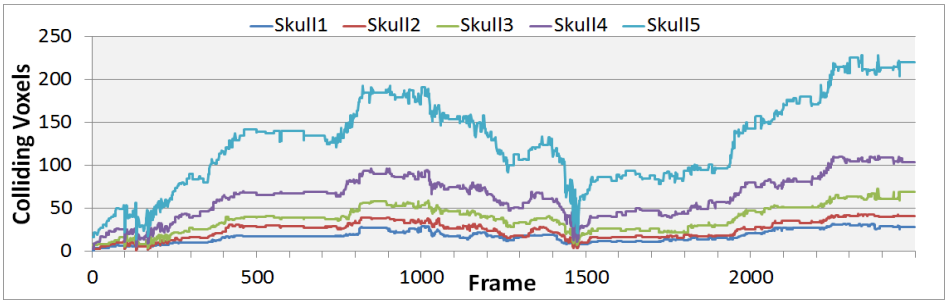
(b) Number of voxels in collision

Figure 3.14: Comparison of the collision times in a non-drilling task for skull models with varying tessellation

The results of the drilling task are shown in Figure 3.15. As expected, the rigid collision handling times are larger when the drilling flag is active than when the tool only touches the skull. The number of colliding voxels increases and consequently the calculation of the force parameters results heavier. Additionally, since the drilling process does not usually require the use of the sweep, the cover of all the colliding voxels is not avoided in most frames. Even though the time consumption of the drilling process is larger than the time spent by the touching task, the collision times only exceed the barrier of $1ms$ when experimenting with the Skull5 model on a few occasions. This means that the rigid collision handling algorithm can be performed obeying the required frequency in most cases. The original skull model contains almost 4000000 voxels, but it could be resampled in order to contain up to 48000000 without the risk of exceeding the claimed maximum collision time.



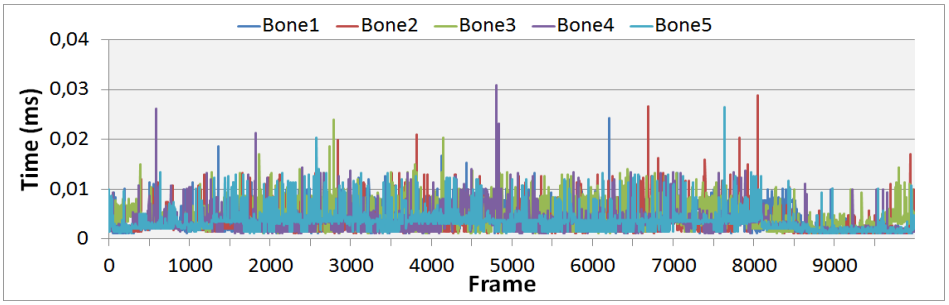
(a) Collision handling times



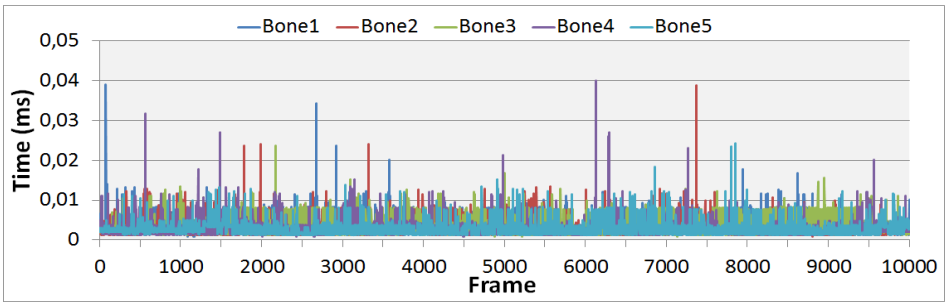
(b) Number of voxels in collision

Figure 3.15: Comparison of the collision times in a drilling task for skull models with varying tessellation

The first section of this chapter has described the rigid volumetric collision handling algorithm designed for the craniotomy simulator. As it explains, the tool's movement is first reflected in the virtual tool and afterwards possible collisions are checked. The movement of the tool is not directly done but a sweep process checks if the entire movement can be executed or some object interferes the whole action. So the system's frequency also depends on the computational cost of the sweep process when moving the tool before handling the collisions. The sweep process combined with the collision handling algorithm must be done in less than $1ms$. We have measured the time consumed by the sweep-before-moving process in 10000 consecutive frames in a scene with skull models. Figures 3.16 (a) and (b) show that the sweeping times are so small that they could even be rejected. The first graph corresponds to the sweep process when touching the skull without drilling. The latter does the same but drilling.



(a) Touch



(b) Drill

Figure 3.16: Times to perform the sweep process for skull models with varying voxel-size for (a) touching the skull and (b) the drilling task

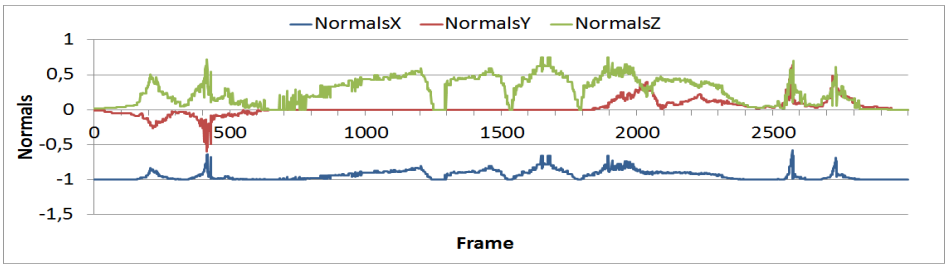
Most of the sweeps are finished in less than $0.015ms$ in both cases. This means that the additional cost of sweeping before moving is absolutely rejectable. The behaviour of all the skull models is similar as the path to cover is usually very small and the voxel size does not nearly affect. In conclusion, our rigid volumetric collision handling algorithm fulfils the frequency requirements of the haptic system even with high voxelization levels. The drilling tasks are computationally more expensive but the minimum frequency of $1kHz$ is achieved anyway.

3.4.2 Force Parameters

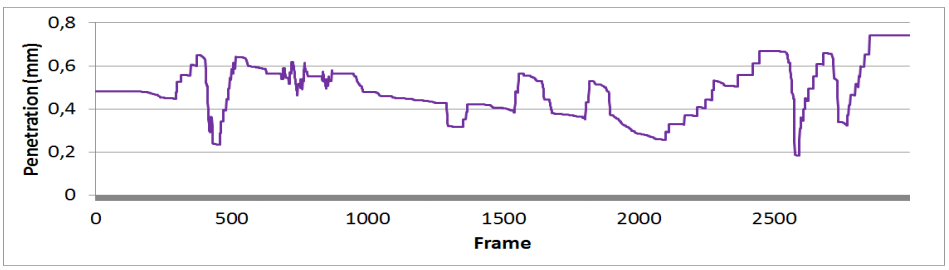
The stability of a haptic system depends on many aspects. Among them, the quantization of the haptic sensors and the saturation of its actuators. However, the haptic interaction model and its parameters are also closely related to the system's stability. It is strongly affected by the force parameters derived from

the collision handling process (the contact normal and the penetration depth) and their variation along time. The contact normal represents the direction of the force restitution to be sent to the haptic device, while the penetration depth directly affects its modulus. For that reason, we have performed some experiments with the aim of determining the behaviour of our method in terms of stability and its limitations.

As done to analyse the computational cost of the algorithm, we have differentiated two tasks the user could carry out in a craniotomy simulator: touch and drill. The former only touches the rigid model but does not drill, while the latter keeps the drilling flag active during the whole experiment. To start with, Figures 3.17 and 3.18 show the force parameters calculated by our rigid collision response algorithm in the touching and drilling tasks respectively. The experiments have been done with the original skull model obtained from the patient's CT. It contains $216x128x142$ voxels of size $0.984x1.584x0.997$. The contact normal and penetration depth information has been measured for 3000 consecutive frames and they represent the force parameters to be included in the final force model.

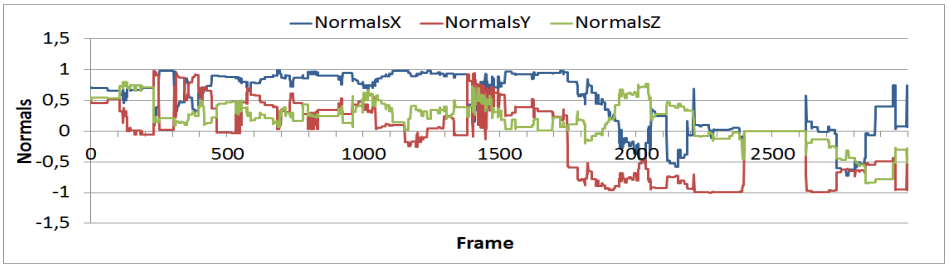


(a) Contact normal

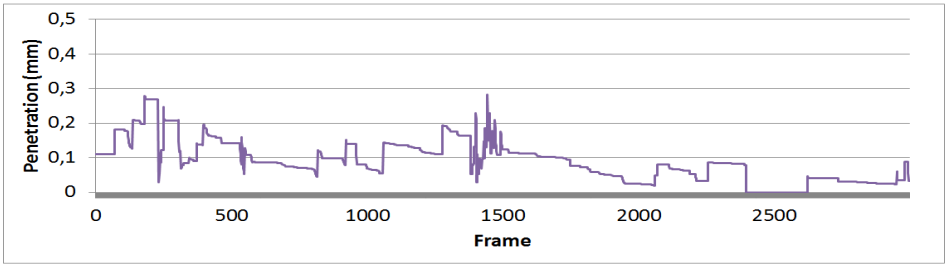


(b) Penetration depth

Figure 3.17: Force parameters returned by our collision handling method in a non-drilling task with the original skull model



(a) Contact normal



(b) Penetration depth

Figure 3.18: Force parameters returned by our collision handling method in a drilling task with the original skull model

Strong variations in the contact normal or the penetration depth can make the haptic device jitter. In the touching experiment, normal directions are reasonably stable and penetrations vary from $0.2mm$ to $0.8mm$ which are quite low values. Knowing that the stiffness value chosen for our system is $k = 2.0$, the force's modulus varies between $0.4N$ and $1.6N$. In the drilling task, the normals seem to suffer more variations, while the penetration depth values are even smaller than in the touching process: they are all below $0.3mm$. This means that having the same stiffness constant than before, the resulting force reaches at most $0.6N$. Since the maximum exertable force of the Phantom OMNI is $3.3N$, the system does not in theory saturate the device in any of the performed tasks. Although large values saturate the haptic device, them themself do not necessarily imply stability problems if they are reached gradually. The next graph shows the variation in time of our penetrations multiplied by the stiffness value ($k = 2.0$). Thus, we can examine the variations in the force's modulus and detect possible instabilities.

As said, Figures 3.19 and 3.20 give the variations suffered by the modulus of the force along time. The absolute value of the modulus difference between

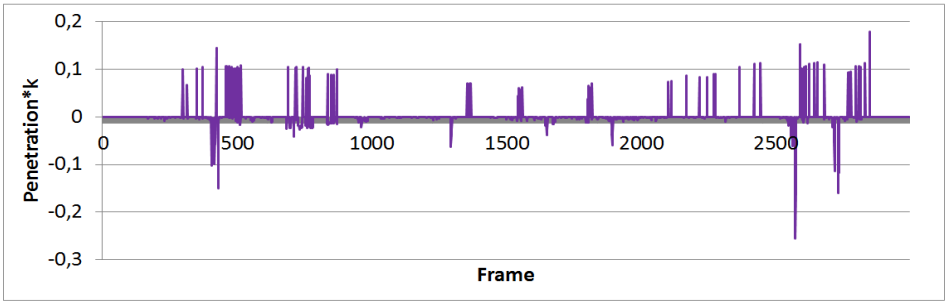


Figure 3.19: Variations in the force's modulus along different frames multiplied by the stiffness constant in a non-drilling task with the original skull model

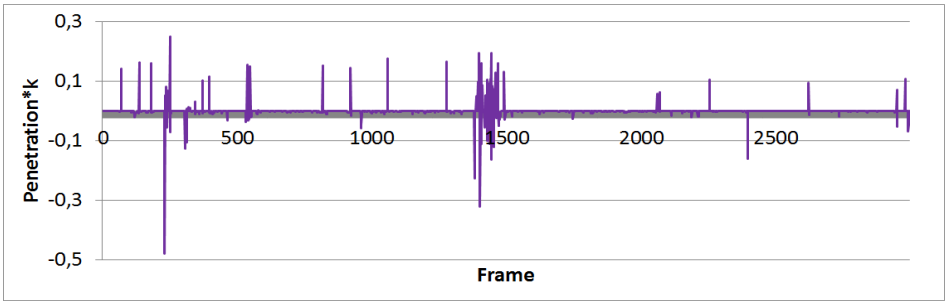


Figure 3.20: Variations in the force's modulus along different frames multiplied by the stiffness constant in a drilling task with the original skull model

two consecutive frames does not exceed $0.3N$ in the touching task and $0.5N$ when drilling. This means that the force stimulus sent to the haptic device does not suffer consecutive variations of more than $0.3N$ and $0.5N$. So in general, with penetration depths beyond $0.8mm$, forces of at most $1.6N$ and force variations that do not exceed $0.5N$, the haptic device is not saturated and the systems seems to be stable.

Our rigid volumetric collision handling algorithm can be understood as an initial calculation of the force parameters with a rectifying stage afterwards. The collision detection module calculates the contact normal and penetration depth based on the current state of the scene objects, but an additional god-object algorithm is later performed in order to manage the position of the proxy tool. In this process, the contact normal and penetration information are rectified based

on the god-object algorithm's results. The information displayed in the previous graphs is the one already rectified. In order to see the initial penetrations calculated by the collision module compared to the rectified ones sent to the haptic device, Figures 3.21 and 3.22 show the penetrations obtained in the previous experiments compared to the penetrations initially calculated by the collision response module.

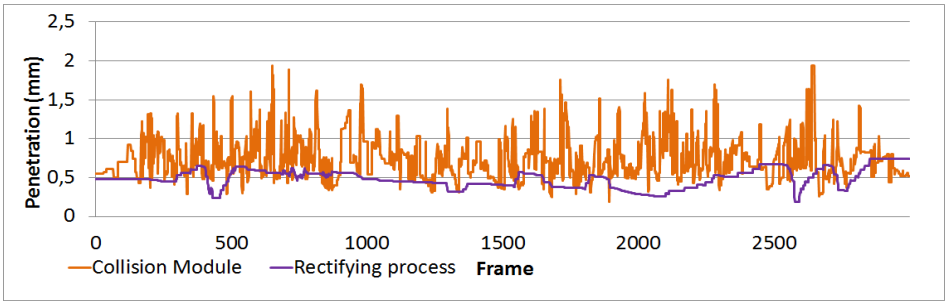


Figure 3.21: Comparison of the first penetrations returned by the rigid collision module and the rectified penetrations in a non-drilling task with the original skull model

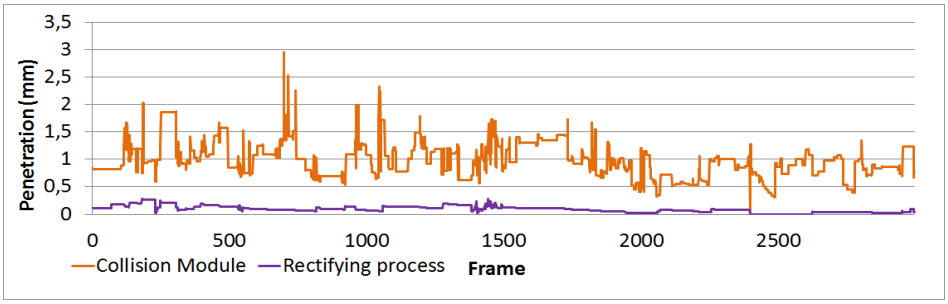


Figure 3.22: Comparison of the first penetrations returned by the rigid collision module and the rectified penetrations in a drilling task with the original skull model

The initial penetration values are rectified based on the god-object location algorithm that calculates the new proxy position. When the god-object algorithm does not need to be run the contact normal and the penetration depth computed in the collision response process are maintained. The graphs show that the rectifying process stabilizes the penetrations sent to the haptic device. In those points where both penetrations coincide, the god-object algorithm has not been activated. The penetrations initially returned by the collision response algorithm are not large

but the rectifying stage makes the even smaller, which results in smaller force restitution values. We could have used a higher stiffness constant in order to obtain a higher force restitution. However, we prefer to maintain low variations in the force instead of receiving a higher force, in order to avoid the risk of getting unexpected high peaks at any time. The visual feedback obtained by means of the sweep process and the god-object algorithm make the user feel a higher force restitution which, together with a sound feedback, completely fulfils the feeling of immersion.

Our rigid collision handling algorithm uses a viscoelastic model to compute the final force restitution to be sent to the haptic device. Figure 3.23 shows the viscoelastic forces returned by our system in the touching task. The force variations in the three axis are also given afterwards (Figure 3.24). Additionally, the values obtained in the drilling experiment are given in Figures 3.25 and 3.26.

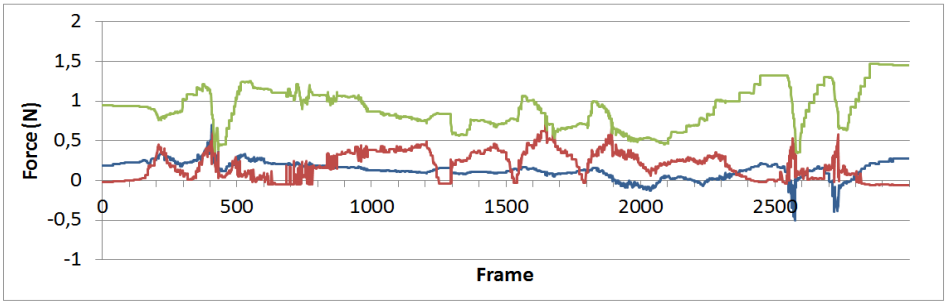


Figure 3.23: Viscoelastic forces in a non-drilling task with the original skull model

As expected, the modulus of the resulting force does not exceed the $1.5N$ barrier which is less than half the maximum force of the Phantom OMNI. With respect to the force variations in consecutive frames, the highest differences in absolute value are all below $0.6N$.

Regarding to the drilling task, the modulus of the resulting force is always smaller than $0.5N$ and the maximum force variation suffered by the haptic device in two consecutive frames is $0.4N$.

In conclusion, he decisions taken along the course of our rigid volumetric collision handling method lead in forces that do not exceed $1.5N$ in any performed task and the force variations in consecutive frames are all lower than 0.4. This means that the force parameters sent to the Phantom OMNI do not saturate the device and vibrations derived from force variations are minimum.

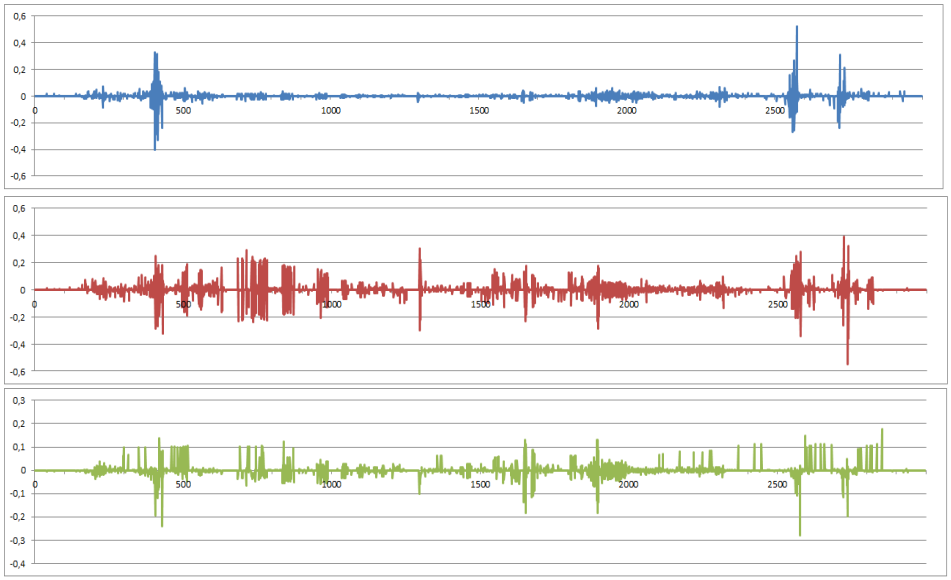


Figure 3.24: Variations in the viscoelastic force along different frames in a non-drilling task with the original skull model

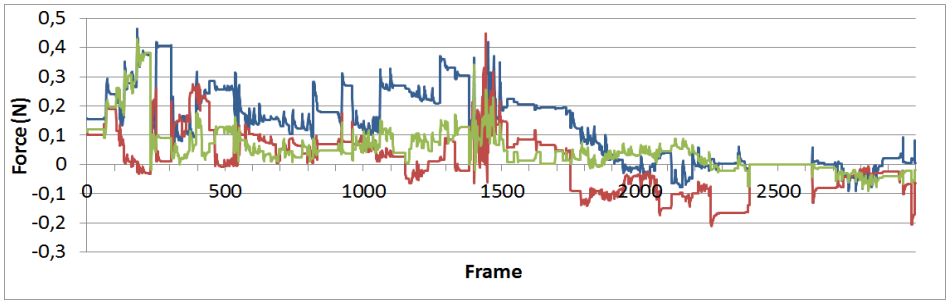


Figure 3.25: Viscoelastic forces in a drilling task with the original skull model

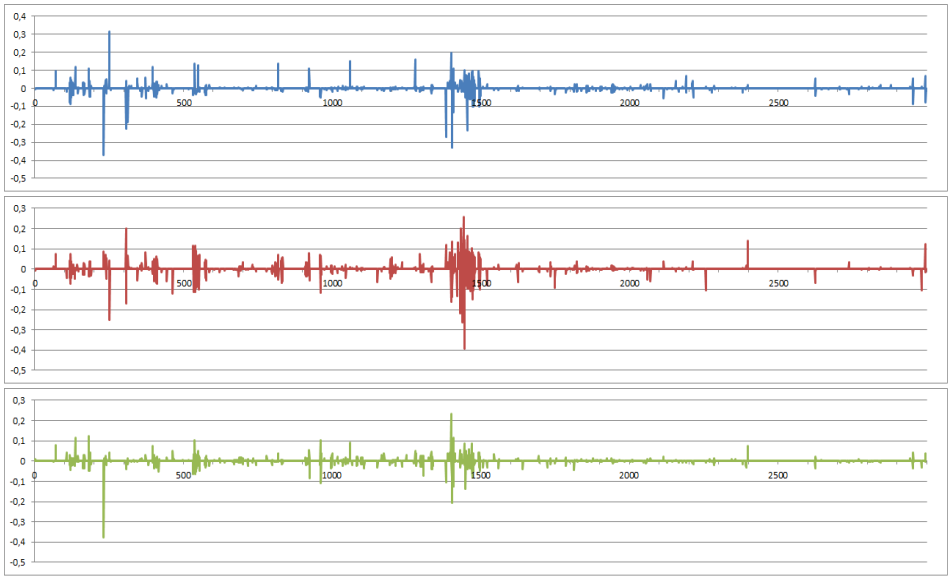


Figure 3.26: Variations in the viscoelastic force along different frames in a drilling task with the original skull model

3.5 Discussion

This chapter has described our rigid volumetric collision handling method to be integrated into a Craniotomy Simulator.

The collision handling includes an accurate collision detection algorithm that detects the interferences between a dynamic virtual tool and a rigid volumetric skull model represented by density voxels. Most of the bone-drilling simulators construct adapted structures from the density-based information in order to avoid the discontinuities inherent to the voxel representation. We have opted to maintain the voxel structure and solve adversities instead of constructing additional structures that can implicitly add errors to our model. In this way, the intersecting primitives (voxels and points) are efficiently determined by only using density data. This information is sent to the collision response module to derive the force parameters that will be used to compute the restitution force.

Thus, the implemented collision response and haptic control method is designed to work with voxel-based information. Based on the colliding voxels and points returned by the collision detection module, the collision response computes

the parameters needed to get the final force restitution which is finally obtained by a viscoelastic force model.

Experiments have been performed for input models with different voxelization levels. The obtained experimental results conclude that our collision detection and response method fulfils the required minimum frequency when touching and drilling bone even with input models containing a big amount of voxels. Moreover, it returns an stable force restitution which is crucial for a realistic and pleasant feeling.

Chapter 4

Deformable Volumetric Collision Handling

If the human brain were so simple that we could understand it, we would be so simple that we couldn't
EMERSON M. PUGH

4.1 Introduction

This chapter describes our approach on deformable volumetric collision handling. This will be integrated into a Brain Haptic Physical Simulator which will be presented in Chapter 6.

The brain interaction requires collision handling for deformable objects. The brain model is a soft body which behaves according to the physical laws, depending on its tissues' properties. For that reason, a collision handling between deformable and rigid bodies is demanded. The main purpose of this work is to get an accurate and realistic haptic interaction with a deformable volumetric brain mesh. The following lines describe the collision detection algorithm performed to solve interferences between a virtual tool and a deformable volumetric brain mesh. The collision response method and haptic control used to transform the geometry information of collisions into a three-dimensional force to be sent to the haptic device will be detailed later. Finally, some experiments will validate our method in terms of time consumption and stability.

4.2 Collision Detection

Collision detection for deformable models adds some difficulties to the one for rigid bodies, which after all can be understood as deformable bodies with infinite stiffness. Deformable bodies can take any form and can move into the space without keeping the centre of mass in a concrete point into the object, unless such movements are restricted by some constraint. Thus, one side of the object can collide with another, which is commonly known as *self-collision* (Magnenat-Thalmann, 1994) (Heidelberger et al., 2004a). Additionally, the deformable nature of soft bodies adds the necessity of updating the data structures every time a deformation happens, while rigid bodies initialize data structures at a preprocessing stage and do not require updates in run time.

Several methods have been applied in the bibliography in order to solve such matters, varying data structures and the representation of the models. Each technique is chosen depending on the environment the algorithm will work on, the data nature and the requirements of the environment.

This chapter presents the collision detection technique we have developed for the interaction of the user with a brain tissue model by means of a haptic device. This method has been tested in a Brain Haptic Physical Simulator, which has later been integrated into a multimodal Neurosurgery Simulator.

Figure 4.1 shows a brief scheme of the developed collision detection algorithm.

Our simulator obtains brain data from MRI images, converting them to an adapted tetrahedral mesh structure. In the same way as in the rigid volumetric collision detection method, the resecting tool is represented by a simple sphere covered by a uniform point cloud that will also be referred to as the *pointshell*.

Hereafter we present the most important procedures contained in the collision detection module.

4.2.1 Initialization

As a preprocess stage, the collision module initializes the tool sphere with its surrounding pointshell, using position and rotation information from the haptic module. The pointshell of the sphere is created in the same way as in the rigid method. However, a different number of points is used. It also initializes all the

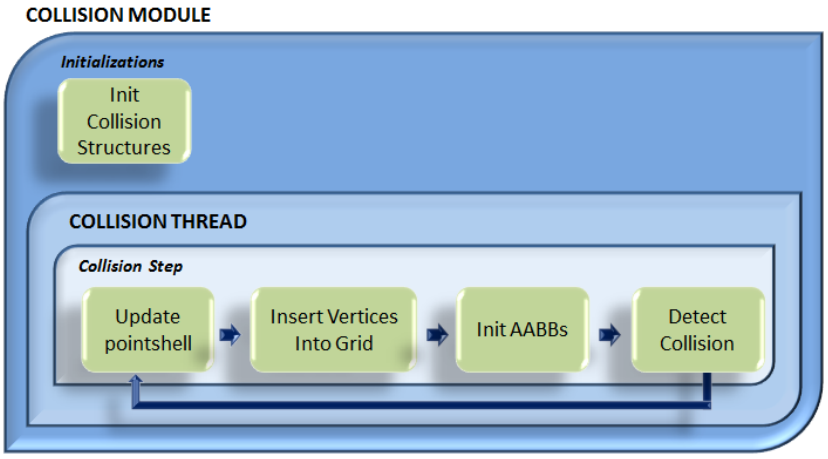


Figure 4.1: Control flow of the collision detection module

collision parameters as normals, penetrations or forces to zero. As mentioned before, the brain model is converted to a tetrahedral mesh. Thus, the axis-aligned bounding box wrapping the mesh is also initialized here.

Since the used collision detection technique is based on a spatial subdivision method, primitives must be placed into the voxel grid to be checked against other primitives. This is made during the collision step process, as it needs to be updated constantly depending on its deformed state. The idea of the spatial subdivision is to divide the space into convex regions called *cells*. These are determined by an identifier (calculated according to the coordinates of the cell), and are placed into a vector in order to classify the interacting objects and determine relations between them.

The technique we have employed uses uniform, rectangular, axis-aligned cells. The structure is based on a three-dimensional array of cells. The size of this array is $N1xN2xN3$, where $N1, N2$ and $N3$ are the number of subdivisions along the respective axes. This structure is constructed at the initialization process. Now the discussion comes when determining the optimal cell-size: as explained before, large cells increase the number of primitives per cell, which takes up a lot of storage and the intersection test slows down. However, very small ones cause a spreading of primitives to more cells which can be too time-consuming. Chapter 5 describes a methodology to infer the optimal voxel size in order to minimize the time-consumption. This methodology has been applied to get the

optimal voxel size for the collision detection algorithm implemented in our Brain Haptic Physical Simulator, so the chosen voxel-size is already optimized.

4.2.2 Collision Step

After the determination of the voxel-size that best fits and the initialization of the data structures, the collision thread starts with the run of the collision step. This is performed once and again until the program finishes. The collision step follows a sequence of four main actions: update the points surrounding the tool depending of the haptic movement, classify them with respect to their new position in the cell grid, update their bounding boxes and finally detect collisions. The four stages are better explained below.

- Update Pointshell: The haptic device moves around its working space and so reflects the virtual tool in the virtual environment. The collision module knows the exact location of the virtual tool constantly so it can determine either a collision is happening or the virtual tool and the brain model are not touching each other. Every time the virtual tool changes its position, the structures related to it are also updated. So the pointshell surrounding the tool sphere is updated to ensure a correct collision detection. This process is simple and fast since it only involves applying the same translation of the tool to each point belonging to the pointshell.

- Classify Points: As explained before, in this technique the space is subdivided into cells. Each cell has an identifier and is introduced into a vector depending on its position in the world. In the same way, all the points of the pointshell can be classified with respect to the cell in which they are placed, and they are consequently placed into an analogous vector. Thus, each cell has a list of references to points contained in the cell.

  The cell access in voxel grids is really fast, since the calculation of the cell where a point in the space lies is made in constant time. For instance, given a point $p(x,y,z)$, the indices of the corresponding cell are:

  $$ind = floor(pos)/cell\_size$$

  for every axis. where *ind* is the indice associated to the cell in one axis, *pos* is the position of the point in that axes and $cell\_size$ is the predefined size of the cell in the same axis.

Once the indices for each cell are known, a function locates them in the vector:

$$id = ind_k \times numCells_x \times numCells_y + ind_j \times numCells_x + ind_i$$

where id is the identifier of the cell in the vector, ind_i, ind_j and ind_k are the indices that locate the cell in the space, and $numCells$ is the number of cells in each axis.

Depending on this function each indice from the vector corresponds to a single voxel from the grid, or if a hash function is applied each indice contains more than one cell. In such a way, storage problems can be solved if necessary. In our case, this is not a critical aspect.

Once all the points belonging to the pointshell are placed into the grid and classified, the algorithm is ready to check them for intersection against other primitives. This will be detailed in the *Collision Detection* section.

- Update AABBs: The pointshell vertices are already placed into the voxel grid. In order to check them for intersection against the primitives of other objects in the scene, these primitives must also be updated and classified. The first step is to update the brain mesh data based on their current deformed state.

  As previously mentioned, our brain model is obtained from MRI data. This information is then transformed into a tetrahedral mesh structure. Those tetrahedra are the primitives used to analyse the intersections between the brain model and the tool. In order to accelerate the collision detection process, the tetrahedra of the mesh have an associated AABB (Figure 4.2). These AABBs are updated every frame based on the current deformed state of the tetrahedron they belong to. The deformation property of the model demands a continuous update of the AABBs since the previous frame could have made changes on them.

- Collision Detection: Once the pointshell is correctly positioned, its points are placed into the grid and the bounding boxes of the tetrahedra are all updated, the collision detection process can be started.

  First of all, equally to the skull drilling method a colliding box is defined that contains the intersection between the axis-aligned bounding boxes of both the brain mesh and the tool-sphere. If the intersection is empty, the

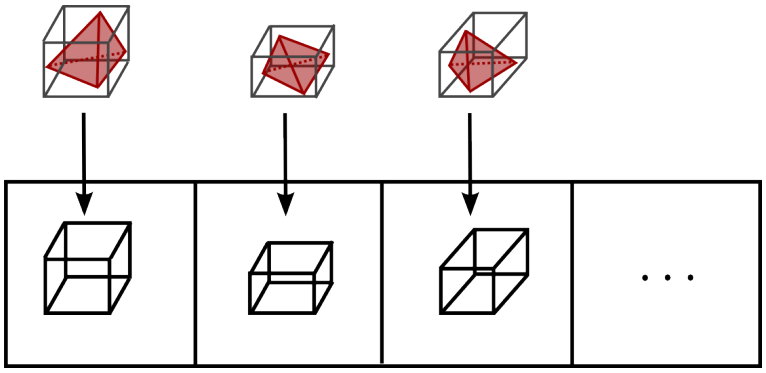


Figure 4.2: The AABBs of the brain tetrahedra are stored in a vector and are updated every frame

collision can be discarded. This first pass avoids checking for interference when the tool is far away from the brain.

If a non-empty colliding box exists, a search along the tetrahedra is carried out. Each tetrahedron is checked for intersection. The search stops when an intersecting tetrahedron is found or when no more primitives are found. Thus, at this point it is already known whether the models are intersecting each other or not. Nonetheless, in the case of collision the process must continue in order to find out more details for the collision response. Data as the contact normal or the penetration depth must be known in order to give a realistic response to the impact. The intersection test is done as in (Teschner et al., 2003).

Intersection Test: The intersection query starts with a tetrahedron to be checked against a group of points belonging to the tools pointshell. The bounding box of the tetrahedron is discretized to determine the cells of the previously defined voxel grid where it lies (see Figure 4.3). For each cell which contains the bounding box of the tetrahedron, the tetrahedron is checked for intersection with the list of vertices contained in the cell.

The intersection test between a vertex v and a tetrahedron t consists of two steps:

- The first one checks v against the bounding box of t, which has been updated in the first pass. If v does not penetrate the bounding box, the procedure stops; there is no collision. However, if v is inside the box a second step is performed.

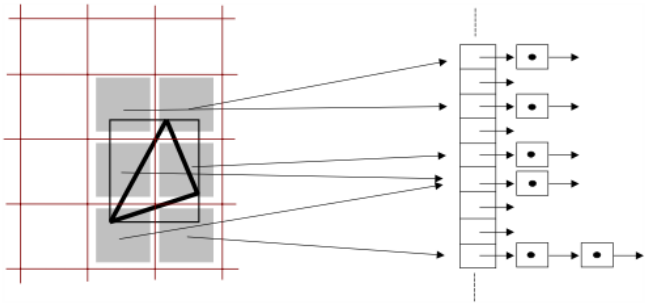


Figure 4.3: The tetrahedron is checked for intersection with the points lying in the same cells as its AABB (Teschner et al., 2003)

– The second step checks whether v intersects t or not. This is made using the barycentric coordinates of the vertex v with respect to the tetrahedron.

  Having the tetrahedron t defined by the points t_1, t_2, t_3 and t_4, the barycentric coordinates of the vertex v in terms of the these points are the numbers α, β, γ and δ such that

$$v = \alpha t_1 + \beta t_2 + \gamma t_3 + \delta t_4$$

  with the constraint

$$\alpha + \beta + \gamma + \delta = 1$$

  Knowing the barycentric coordinates of the vertex v with respect to the tetrahedron t, v is inside or part of the tetrahedron t if and only if

$$0 \leq \alpha, \beta, \gamma, \delta \leq 1$$

If the second stage finds an intersection between the point and the tetrahedron, the collision has already been detected.

The diameter of the tool in our simulator is substantially smaller than the facets of the tetrahedra. That is why if a collision exists and if one of the intersecting tetrahedra is located, checking some tetrahedra around it instead of covering all the primitives again would be enough. What we do is to analyse neighbours of the found tetrahedron, as well as the neighbours of these neighbours. The rest of the tetrahedra are directly discarded. Figure 4.4 shows the tetrahedra checked for intersection with the virtual

tool. Those tetrahedra are undergone the same collision detection method mentioned before. This stage substantially decreases the time consumption of the algorithm which is crucial when integrating the method in a haptic system.

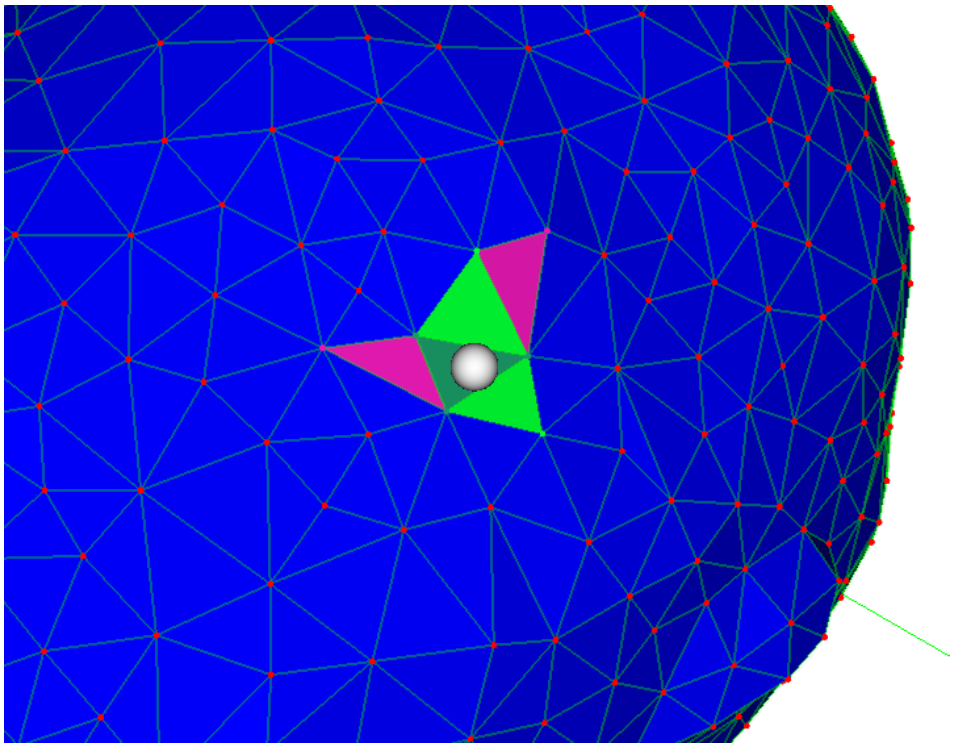


Figure 4.4: Tetrahedra to be checked for intersection with the virtual tool

Once the suspicious tetrahedra have been analysed, it is already known whether a collision has happened or not. What is more, the colliding tetrahedra and the points inside each of them have been located. This information is afterwards used to calculate the contact normal and the penetration depth, which will be performed in the collision response stage detailed in the next chapter.

As previously stated, this four steps are run once and again by the collision thread. For every frame, the information of the contact normal and the penetration depth is available and the haptic module can use it to calculate the resulting

force feedback to be sent to the haptic device. The next chapter explains the collision response algorithm used for both the rigid volumetric and the deformable environments.

4.2.3 Visual Feedback

Apart from the force feedback, a visual feedback is also required. As explained in the rigid volumetric collision handling algorithm, the haptic interface is not infinitely stiff, so the movements of the user cannot be completely restricted. Even though the collision handling method returns a force feedback, the force limitations of the haptic device must be overcome in some way.

The deformable collision handling method requires a less sophisticate visual feedback than the rigid volumetric collision handling method, since the brain tissues have a lower stiffness than skull bones. The deformable collision handling algorithm also claims a visual feedback, but since the haptic interaction with the brain is thought to be slight and smooth, the visual feedback does not require much. What we do is to apply the same god-object algorithm as in (Zilles and Salisbury, 1995a). Given the location of the virtual tool, the proxy position and a restriction plane, the algorithm returns a new plausible position for the proxy. Our collision detection method, after performing the four stages described above, already has enough information to define the restriction plane claimed by the god-object method (it is derived from the contact normal and the previous proxy position). In this way, once the collision handling algorithm calculates the collision information for a frame and if a collision is happening, the system computes the new proxy position by means of the god-object method and the visual representation of the tool is relocated in the brain surface (Figure 4.5). In the case of no collision, the proxy position is matched with the real haptic position.

4.3 Collision Response and Haptic Control

It has already been stated that the visual feedback received by the user could be enough to perform the surgery tasks as skull drilling or interacting with the brain tissue. Nevertheless, adding a force stimulus that restricts the movements of the user along the interaction with the virtual environment strongly increases the accuracy of the application.

As said before, the final force computation depends on the collision

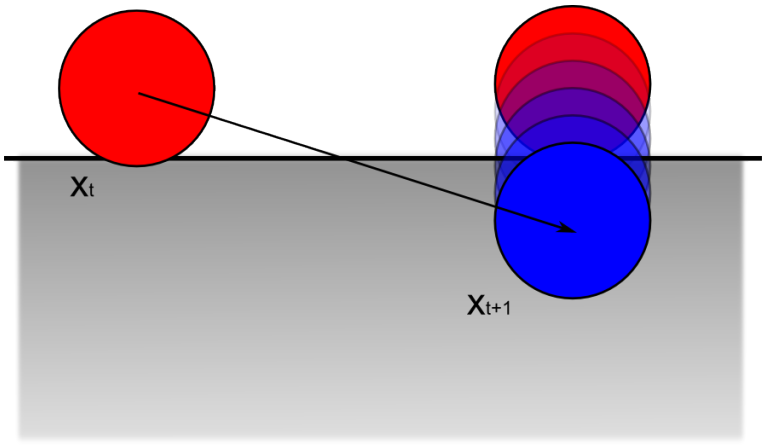


Figure 4.5: New proxy position obtained from the god-object method

information obtained from the collision detection process. In the case of the rigid collision detection, the given information is the list of colliding voxels and the tool points interfering each voxel. In the case of the deformable brain collision detection, since the brain model is represented with tetrahedra, surface information is available. This will be used to compute the force feedback for our deformable brain collision handling approach.

Figure 4.6 illustrates the control flow of the whole deformable volumetric collision handling process. As it is shown, the collision response module calculates the final force to be sent to the haptic device based on the data received from the collision detection module. The collision response module calculates the final force to be sent to the haptic device based on the information received from the collision detection module. This module constantly examines the situation of the scene objects and gets the colliding primitives. The collision response module uses this information to compute a set of three-dimensional forces that act against the restrictions.

The deformable collision detection module finds interferences between the scene objects and obtains some data related to such collision: colliding tetrahedra and the points intersecting each colliding tetrahedron. These data are used to calculate the force restitution to be applied to the haptic device and the brain nodes in order to simulate the proper deformation.

The contact normal is defined as the average of the normalized surface face normals belonging to the colliding tetrahedra. Each tetrahedron contains the list

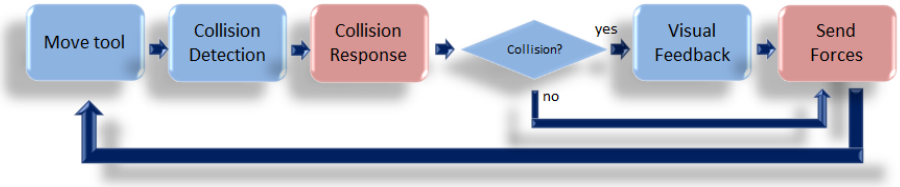


Figure 4.6: Control flow of the deformable volumetric collision handling process

of its neighbours, so the surface faces are easily identified as the ones with no neighbours.

Concerning to the penetration depth, it is calculated as the maximum of the projected distances between the colliding points and the surface faces of the tetrahedra they belong to. Algorithm 2 shows the pseudocode of the penetration depth calculation, and Figure 4.7 illustrates it graphically.

Algorithm 2 Deformable penetration depth

```
1:  function DEFORMABLE PENETRATION(tetrahedra, points)
2:      dist ← 0
3:      penetration ← 0
4:      for i = 0 → tetrahedra.size() do
5:          for j = 0 → points[i].size() do
6:              for k = 0 → surfaceFaces(tetrahedra[i]) do
7:                  dist ← abs(ProjectedDistance(points[i][j], surfaceFaces[k]))
8:                  if penetration < dist then
9:                      penetration ← dist
10:                 end if
11:             end for
12:         end for
13:     end for
14: end function
```

With all this information, the system again returns a three-dimensional force. It already has the necessary information to compute the force restitution: the contact normal and the penetration depth. With the aim of giving a more realistic feeling to the user, the force model in this case also adds a viscous component to the elastic model defined previously. So the force model used is the viscoelastic model described in Equation 3.3.

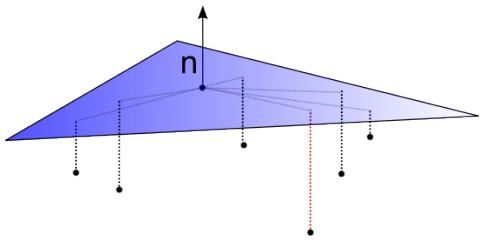


Figure 4.7: The penetration depth is the maximum of the projected distances between the points and the surface facet

The choice of the force model for each case has been made experimentally. Different models have been applied to both simulators, taking the most realistic ones. In this case, having a deformable brain model, damping makes the user feel a padded model of the brain giving a richer perception.

The proposed Craniotomy and Brain Haptic Physical Simulators require different collision detection and response techniques. The first one demands a method to manage interferences between the virtual tool and a rigid volumetric object, while the second one requires deformable collision handling techniques. Both detection modules return information related to the interferences produced against a virtual tool governed by the user. This information must then be processed to send the pertinent force feedback to the haptic device. The deformable nature of the brain model prompts a topology change when an external objects gets in touch with it. Hence, the appropriate force is also sent to the nodes of the brain model affected by the interference. Thanks to the collision detection of the rigid volumetric skull and the deformable brain model, the necessary information to compute a plausible and realistic force feedback has been acquired.

4.4 Experimental Results

Haptic systems have the goal of avoiding discontinuities that lead in an unpleasant feeling for the user. The proper models are represented by discrete structures that implicitly add discontinuities, so the problems start from there.

What is more, the time consumption of the algorithms is also vital in order to offer an accurate response. Haptic systems usually require a minimum frequency of $1kHz$ to offer a real-time interaction. The collision module is forced to finish its work in less than $1ms$ to achieve the mentioned frequency. Otherwise, the haptic

device receives force information that does not correspond to the real situation of the scene. This effect must be avoided by forcing the collision handling algorithm to fulfil the claimed frequency.

As we did to evaluate the continuity and stability of the rigid volumetric collision handling method, the deformable volumetric collision handling algorithm has been examined by analysing two main aspects: the time consumption of the collision handling and the stability of the force parameters.

The computational cost of the collision handling method will show whether the minimum frequency can be achieved or not and the factors that most affect it. We want to analyse the time consumption differences between input models with varying tessellation. Since the idea is to compare the behaviour of an algorithm depending on the used models, an identical path is used. The same tool path is covered for the same algorithm with different models. In such a way, the comparison is done by only varying the parameter we are owing to compare. This path has been recorded in the same way as we did for the rigid method experimentation.

It has already been said that the number of points contained in the pointshell also affects the time consumption. However, what we want to do is to analyse the tessellation of the input models instead of the tool's point density. In the deformable collision handling case, the chosen step for α and β is 15.0, which leads in 211 points surrounding the sphere. This choice is the one that best time-accuracy balance offers. This value is maintained for all the deformable experiments.

Once again, the first experiments are done with cubical models which help us to identify the possible bottlenecks of our algorithm. The time consumption of our method with some brain models will be given later.

With regard to the system's stability, the force parameters returned by the collision response algorithm will be given.

4.4.1 Time Consumption

The deformable collision module's frequency depends on basically two factors: on one hand, the number of primitives of the brain model significantly affects the collision detection's time consumption. A big amount of tetrahedra entails a large number of intersection tests, which increases the detection time. On the other hand, as will be stated in Chapter 5, a bad election of the voxel size can

substantially slow down the work of the collision module. The latter has already been optimized by a methodology presented in this thesis. So the experiments have been performed using the best fitting voxel size for each model.

To start with, Figure 4.8 displays the execution times of the deformable collision handling algorithm with different cube models, using a cell-size that according to the previously mentioned study optimizes the calculations for each case. The tessellation level and the chosen cell-size for each model is described in Table 4.1. The pointshell surrounding the tool contains 211 points.

model	tetrahedra	cell-size
Cube1	1715	37.5
Cube2	10985	20.0
Cube3	53240	12.0
Cube4	135000	8.7
Cube5	320000	6.5
Cube6	625000	5.2
Cube7	1080000	4.2

Table 4.1: Description of the cube models and the chosen cell-size

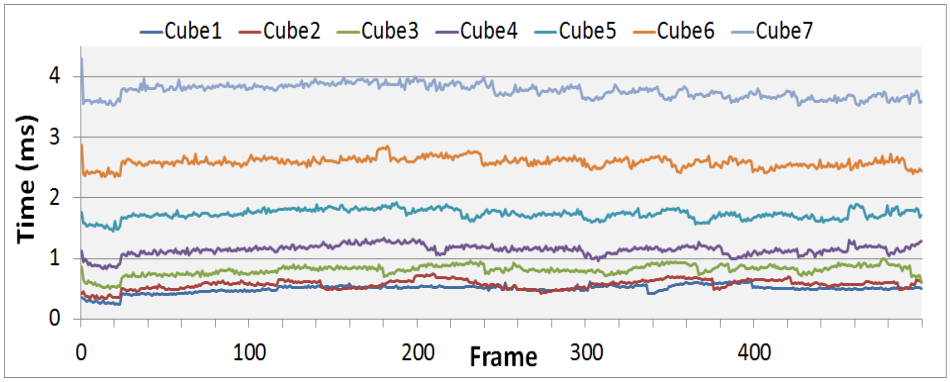


Figure 4.8: Comparison of the collision times for cube models with varying tessellation

It must be noted that outliers can appear in the graphs since the analysed time values are really small and any unexpected process run in the computer can significantly distort the values. For the models Cube1, Cube2 and Cube3 all the values lie inside the 1ms barrier so even the outliers are not a problem. For these three models the required minimum frequency is achieved and thus

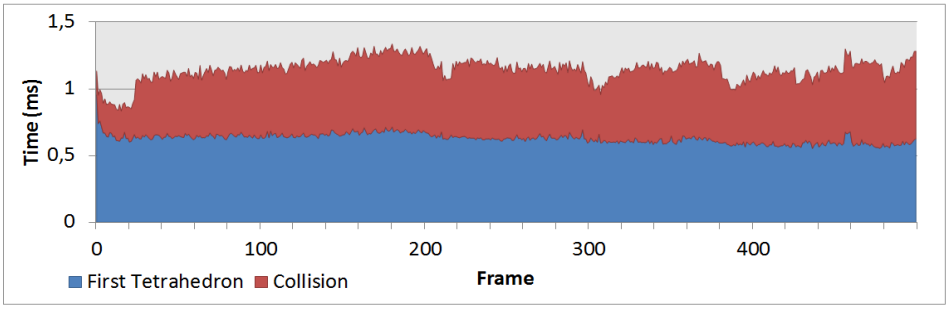
the system is stable. The number of tetrahedra of the first model is quite small, so the collisions are fast but the required realism could be affected. The second and third models have 10985 and 53240 tetrahedra respectively, which apart from providing an stable collision handling offer the required minimum precision. The main discussion comes when deciding whether these tessellations are sufficient in order to approach the demanded realism. Concerning to the models Cube4, Cube5 and Cube6, their execution times pass the barrier. The collision algorithm should be better analysed to detect the real reason of such alterations.

The collision detection algorithm is constituted by different phases, and analysing the times consumed by different stages could help us understand where the real bottleneck is. The method starts with the search of the first colliding tetrahedron. Afterwards, its neighbours with index 2 are examined and the rest of the algorithm is performed. Therefore the collision times can be divided in order to find out the most time consuming part of the algorithm. The graphs Figure 4.9 (a) and (b) show the collision handling times for the Cube4 and Cube5 models, separated in two stages: the search of the first colliding tetrahedron and its neighbours is illustrated in blue while the rest of the algorithm is given in red, stacked.

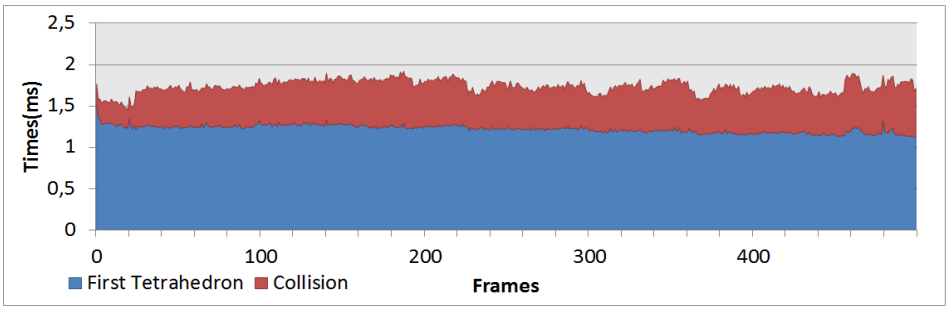
The results suggest two things: first of all, in the case of exceeding the 1ms barrier, the most time consuming stage seems to be the search of the first colliding tetrahedron and its neighbours. Let's better see the same times but independently shown in order to see the differences between both stages (Figures 4.10 (a) and (b)).

Clearly, the bottleneck is in the search of the colliding tetrahedra. The rest of the calculations are all done in less than the haptic module claims. Now, the question at this point is: which are the parameters that affect the search of the first colliding tetrahedron? Does the tessellation level influence it? Well, it actually does. It can be seen in Figure 4.11 where a comparison of the times consumed by different cube models to compute the search of the first tetrahedron and its neighbours is displayed.

It is obvious that the more tetrahedra the model contains, the more time it needs to do the search. However, the location of the collision is also a vital aspect. The search of the first colliding tetrahedron is done starting from the first stored tetrahedron. Without any other criterion. For that reason, if the collision is located near the first searched primitive, the time consumed to find it is small. On the contrary, if the collision is happening at some tetrahedra at the end of the primitives' vector, the time spent to reach them is longer. Experiments with real



(a) Cube model of 135000 tetrahedra and a cell size of 8.7mm

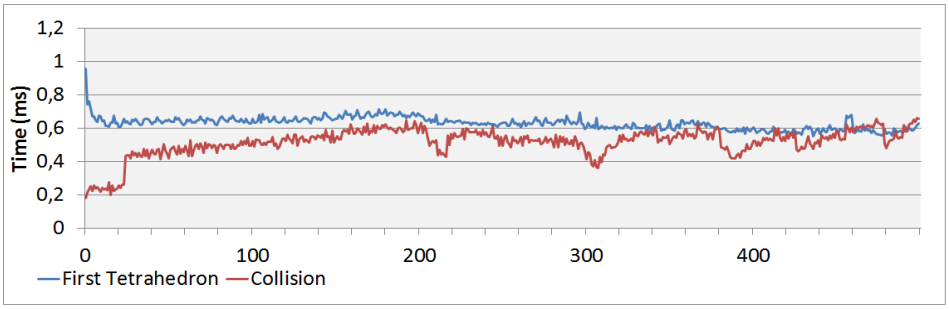


(b) Cube model of 320000 tetrahedra and a cell size of 6.5mm

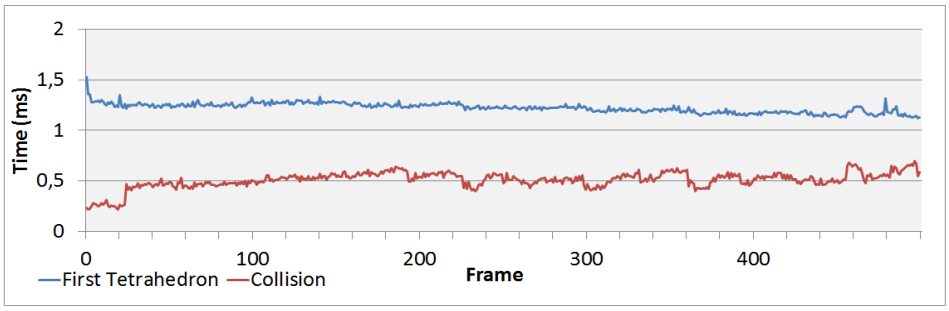
Figure 4.9: Times to perform the collision handling process for Cube4 and Cube5 divided into two steps: the search of the first colliding tetrahedrons with its neighbours and the rest of the algorithm

brain models will confirm the necessity of improving this stage.

The deformable collision handling method implemented in this thesis is thought to be used in a neurosurgery simulator, so the deformable input models used are brain models. Their tessellation is between 10000 and 20000 tetrahedra aiming for a compromise between approximation to reality and stability. Apart from that, the simulation module is not able to support a larger amount of primitives in real time. The first time consumption experiments have been done with cube models that contain until 50 times more primitives. This serves to find out the problems our collision handling algorithm itself could have. Let's now see the time consumption for different brain models. Once again, it must be pointed that although the compromise between realism and stability is best achieved with models Brain2 and Brain3, the experiments have been done with other tessellation levels in order to exalt the differences.



(a) Cube model of 135000 tetrahedra and a cell size of 8.7mm



(b) Cube model of 320000 tetrahedra and a cell size of 6.5mm

Figure 4.10: Times to perform the collision handling process divided into two steps for Cube4 and Cube5

Table 4.2 describes the used brain models with their tessellation and the chosen cell-size that best fits them. Figure 4.12 illustrates the comparison of the times consumed by the collision handling algorithm for the mentioned brain models.

model	tetrahedra	cell-size
Brain1	3497	15.5
Brain2	12511	10.5
Brain3	22076	8.6
Brain4	40584	7.0
Brain5	50041	6.4

Table 4.2: Description of the brain models and chosen cell-size

Brain1 and Brain2 are performed in the time required by the haptic module.

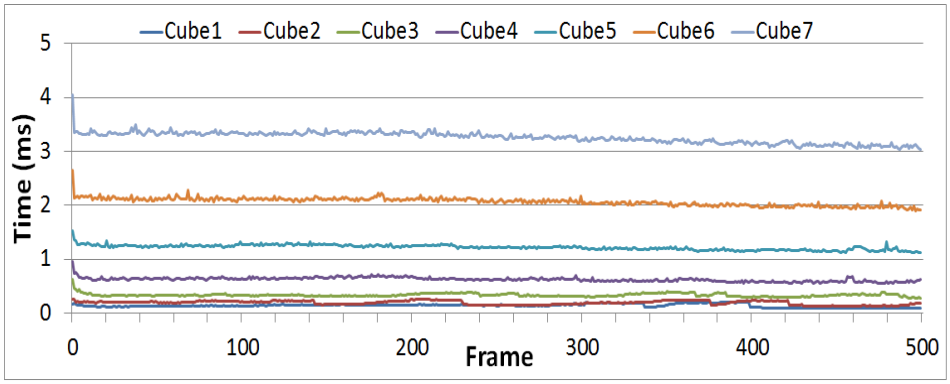


Figure 4.11: Comparison of the times to perform the first stage of the collision algorithm for cube models with varying tessellation

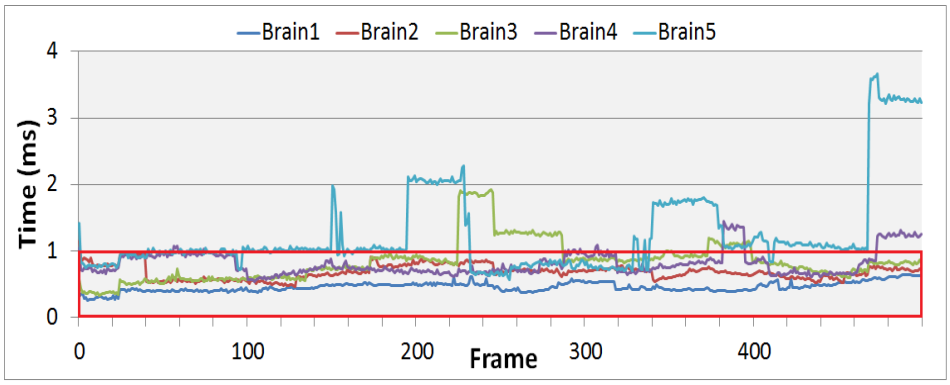


Figure 4.12: Comparison of the collision times for brain models with varying tessellation

On the contrary, the collision algorithm needs more than 1ms to finish its calculus in some frames when loading brain models that contain more than 20000 tetrahedra: Brain3, Brain4 and Brain5. For the models Brain4 and Brain5, some time-exceeding ranges can be found. This means that for some collision zones, the times needed to compute the collisions does not achieve the minimum frequency. The case of Brain4 is not so worrying as even if it sometimes passes the claimed time limit, the excess is minimal.

As said before, the haptic thread requires a minimum frequency of 1kHz. In other words, the collisions must be performed in less than 1ms. This is the ideal case. Otherwise the haptic module does not receive collision information during

some frames and it must activate mechanisms to predict the force parameters. In our simulator, when the haptic module does not receive any information from the collision module, the force parameters received in the last frame are maintained. In this way the force stability is not altered. When the time consumed by the algorithm is smaller than 2ms, the haptic thread only needs to maintain the values for one additional frame. Nonetheless, if the time consumption is bigger, the maintenance prolongs in time and the force feedback felt by the user does not correspond to reality. Additionally, if the time excess happens in consecutive frames, the system suffers big instabilities and the force and visual feedback are not pleasant for the user.

The brain models used in our simulator have a similar nature to Brain2 and Brain3. Although the former satisfies the frequency requirement, the latter does not. As mentioned above, an improvement can be done in the search of the first colliding tetrahedron in order to reduce its computational cost. The first colliding tetrahedron is reached by covering the primitives' vector starting from the first stored tetrahedron. For that reason, if the collision is by chance located at the beginning of the covered vector, the time consumed to find it is small. Contrarily, if the collision happens at some tetrahedra at the end of the primitives' vector, the search results much longer.An easy way to reduce this time is assuming that for consecutive frames the collision will be happening in close tetrahedra. So the first search could result heavy, but the rest would be started from tetrahedra around the last collision location. In such a way, the search of the first colliding tetrahedron could be significantly reduced in many points. In sum, the search is started from the location of the previous collision, so in those cases where the collision is located in near tetrahedra the time consumption is significantly reduced. The new obtained times are shown in Figure 4.13 for all the brain models seen before.

The graph shows that the ranges where the non-improved algorithm gives high collision times are reduced to a single frame with a high time value. The ranges where the time consumption exceeds the required time are caused by collision zones located far away from the beginning of the primitives' vector. Thanks to the new algorithm, once a collision has been found, the next search is started from the found collision zone. The previous graph demonstrates that the collision times are significantly reduced and in the cases where the times pass the barrier, the excess is not maintained in time.

The following figures compare the values corresponding to the most time-consuming brain models from Figure 4.12 and Figure 4.13, but only measuring the first stage of the algorithm. Figure 4.14 shows the times consumed

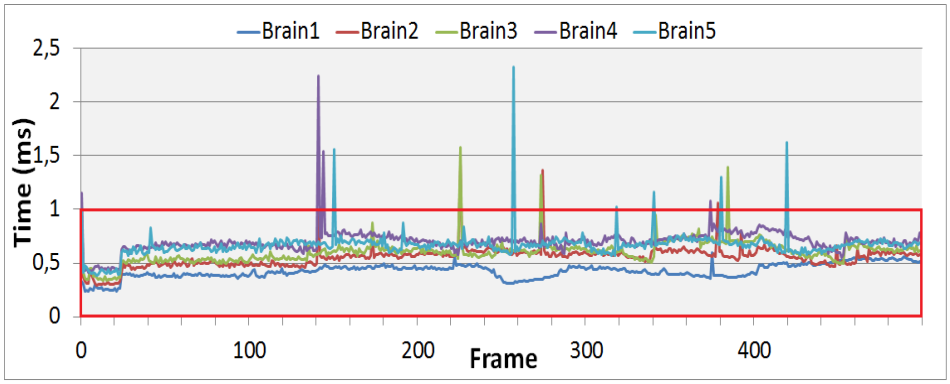


Figure 4.13: Comparison of the improved collision times for brain models with varying tessellation

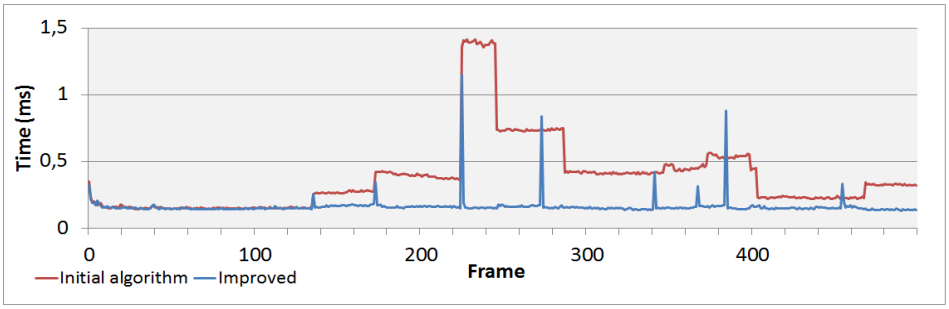
by the Brain3 and Brain5 models to finish the first stage of the collision algorithm with both methods: the original one and its improved version.

The second version of the algorithm evidently reduces the most time-consuming ranges to a single outlier which, for the case of Brain3 does not even exceed the 1ms. The system is now much more stable, and in the cases where the time consumption does not fulfil the claimed time limits, the excess is not prolonged in consecutive positions.

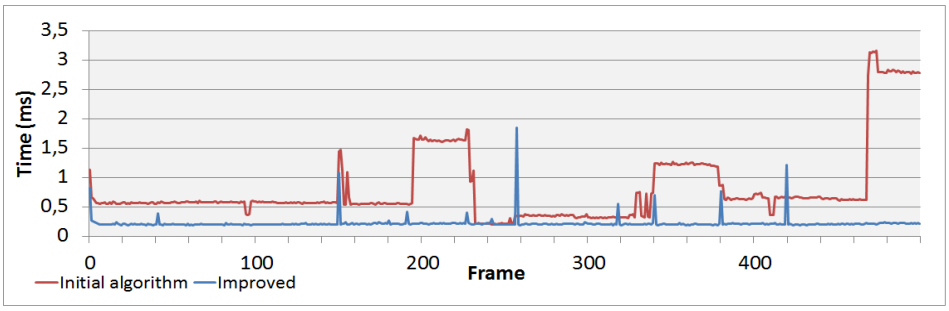
4.4.2 Force Parameters

Although some of the stability limits of a haptic system depend on the sensors quantization and the saturation of the actuators, most of them can be obtained studying the haptic interaction model and its parameters. The stability of a haptic system largely depends on the force parameters derived from the collision detection process: the contact normal and the penetration depth. They represent the direction and module of the force restitution which is crucial to ensure a stable response of the haptic device.

This section displays a set of experiments carried out with the goal of determining the stability of our method in different situations: moving the tool around the surface of the brain, a freehand random movement, and a localised interaction in a concrete zone with higher intermittent penetrations. The latter recreates the movements of a surgeon when interacting with a concrete brain zone. Table 4.3 describes the three paths used to perform the experiments. We analyse



(a) Brain model of 22076 tetrahedra and a cell size of 8.6mm



(b) Brain model of 50041 tetrahedra and a cell size of 6.4mm

Figure 4.14: Comparison of the times to perform the first stage of the collision handling process for Brain3 and Brain5, with the initial search algorithm and its improved version

the normals and penetrations returned by the collision module in consecutive frames of such paths with different brain models.

Figures 4.15, 4.16 and 4.17 show the normals and penetrations returned by the deformable collision module during 3000 frames in a scene with a brain model of type Brain3 by covering the mentioned three paths.

Strong changes in the contact normal or the penetration can make the haptic device jitter. However, large penetration depths themselves do not imply system instabilities if they are reached gradually. Having a look at the first graph (Path1), at first glance it suggests that when a hard change of the normal direction happens (see frames $0-400$ and $2900-3000$) the penetration depth grows or decreases gradually. This leads in a smoother feeling and avoids high force peaks. Nonetheless, only with the shown normal vector and penetration information it is not easy to assure that the force peaks are not excessive. The results must be better

Path name	Description
Path1	Slight movements covering the surface of the brain
Path2	Freehand random movements through the surface varying slight and strong contacts
Path3	Recreation of the movements a surgeon would do to interact with a localised zone of the brain

Table 4.3: Description of the different paths recorded for the experiments

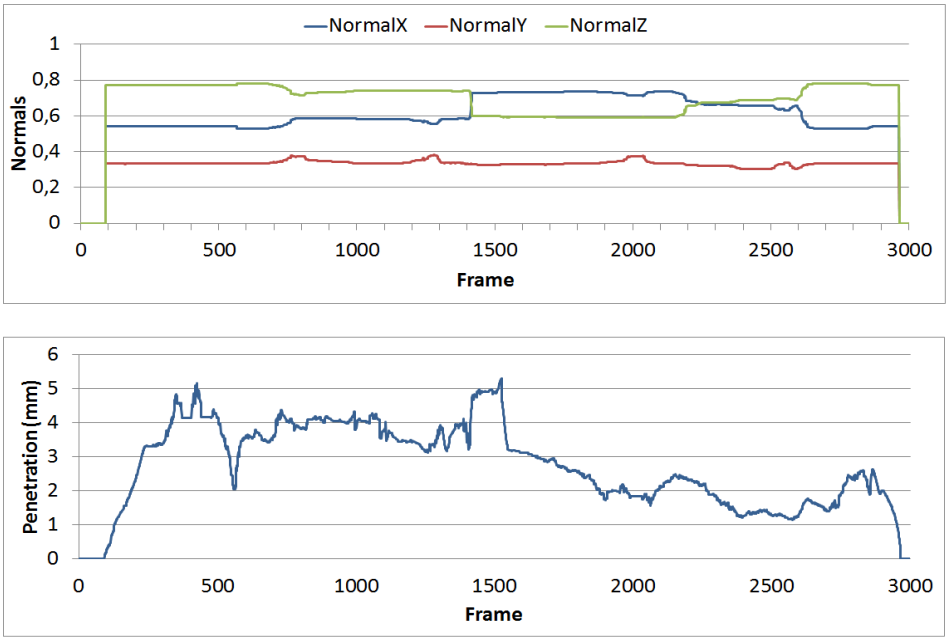


Figure 4.15: Force parameters returned by the deformable collision handling algorithm for a Brain3 model and responding to Path1

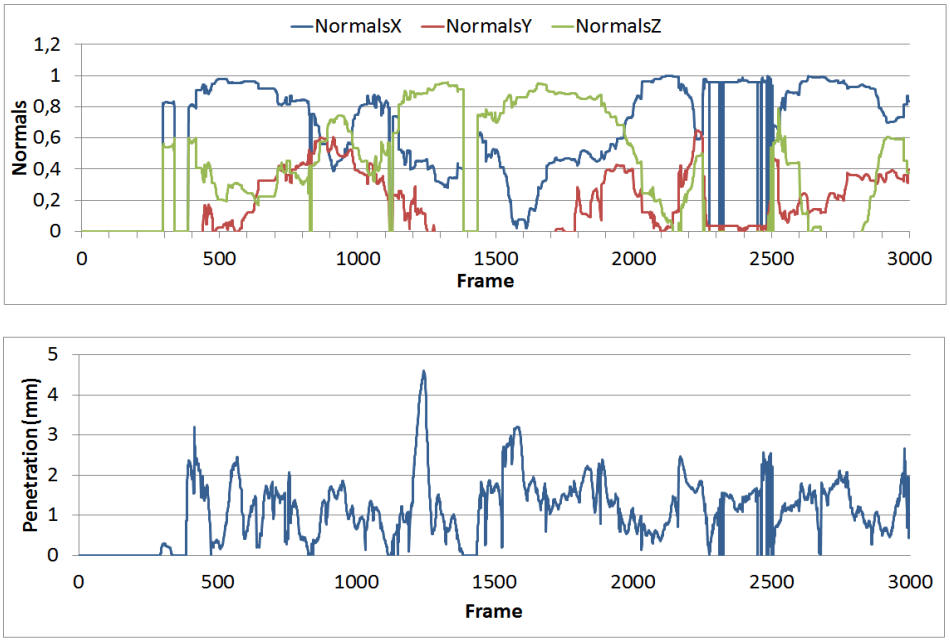


Figure 4.16: Force parameters returned by the deformable collision handling algorithm for a Brain3 model and responding to Path2

examined.

As mentioned before, we have analysed three different paths in order to examine the stability of the algorithm in different situations. Figure 4.18 gives the penetration depths obtained for the three cases with the Brain2, Brain3 and Brain4 models.

As might be expected, when the tool moves around the brain surface lightly the penetration depths do not vary pretty much. However, in the case of freely and randomly moving around the environment produces strong variations in the penetration depth and causes undesirable peaks. The third experiment, where the surgeon's brain interaction is reproduced, returns a kind of wave of penetration depths as a result of touching the same zone of the brain once and again. Nevertheless, as said before, the penetration depth itself does not inform pretty much about the system's stability so its variations along consecutive frames will be analysed.

The viscoelastic model used to calculate the final force sent to the haptic

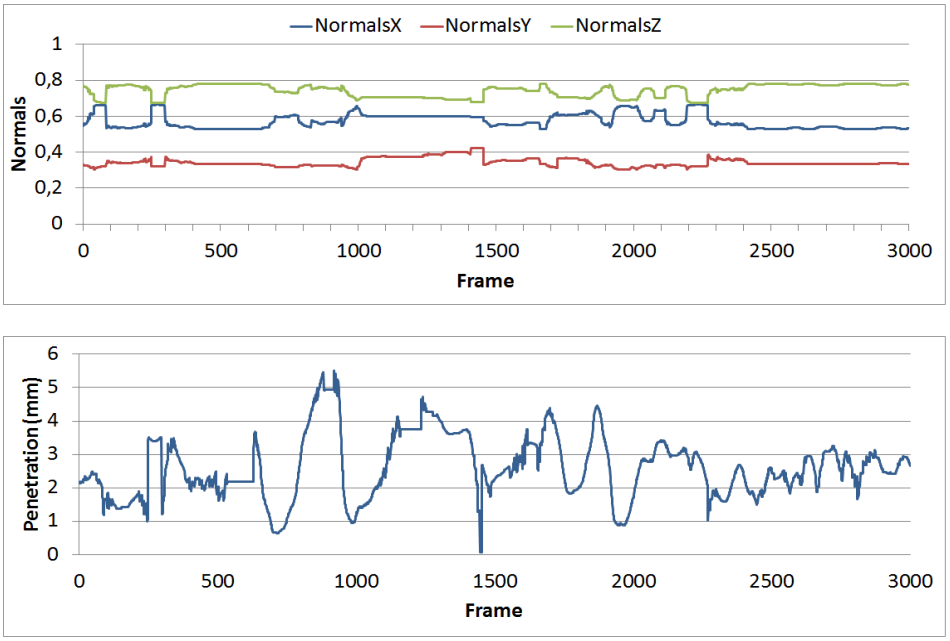
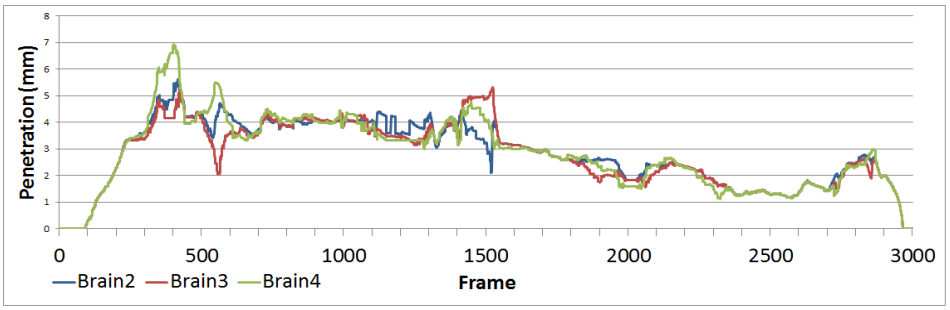


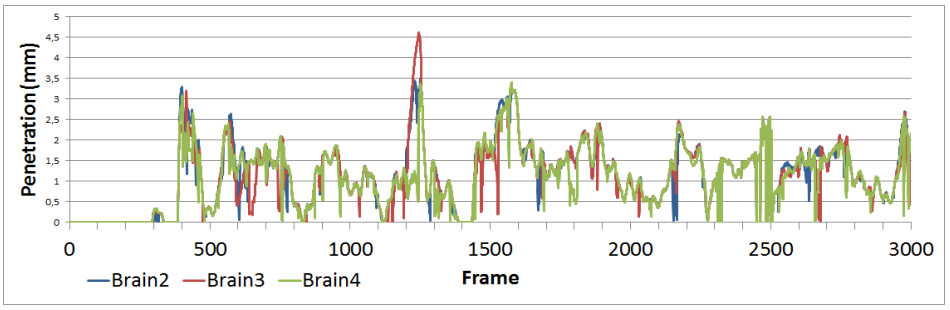
Figure 4.17: Force parameters returned by the deformable collision handling algorithm for a Brain3 model and responding to Path3

device is based on the contact normal, the penetration depth, the stiffness constant, the damping constant and the velocity. In order to better analyse the system's behaviour we have multiplied the penetration depths of Brain3 obtained in the previous graphs with the stiffness constant (in our case $k = 0.3$), and Figure 4.19 (a) gives their variation between consecutive frames covering Path1. (b) gives the variation in the freehand experiment (Path2) and (c) is the variation returned by Path3: localised collisions. The damping constant and velocity have not been applied in the experiments as they can be understood as an additional contribution used to smooth the resultant force. The penetration difference multiplied by the stiffness constant shows the variation on the resultant force if a simple elastic model was applied. The effect of applying a viscoelastic model instead of an elastic one will be displayed later.

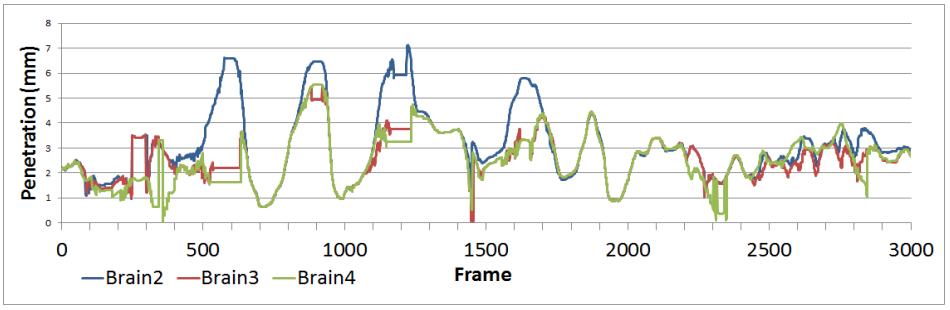
The results show that the variations suffered by the haptic device are minimal in all the analysed situations. The biggest peaks are of up to $0.8N$, which would carry a force variation of only 0.8N. Exceeding the maximum exertable force of the haptic device usually causes large vibrations. Nonetheless, taking into



(a) Path1: moving the tool around the surface

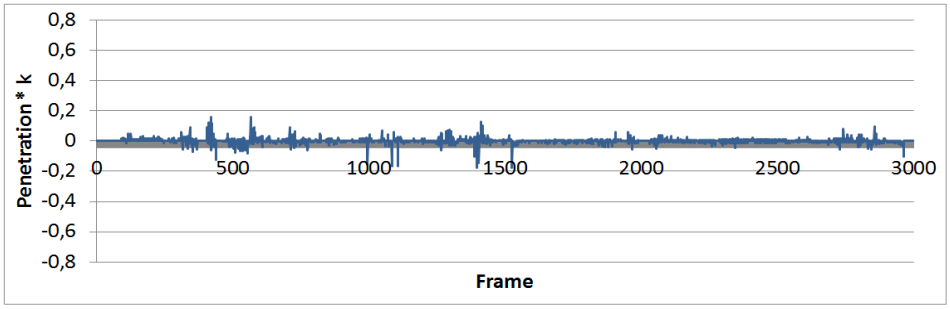


(b) Path2: Random freehand experiment

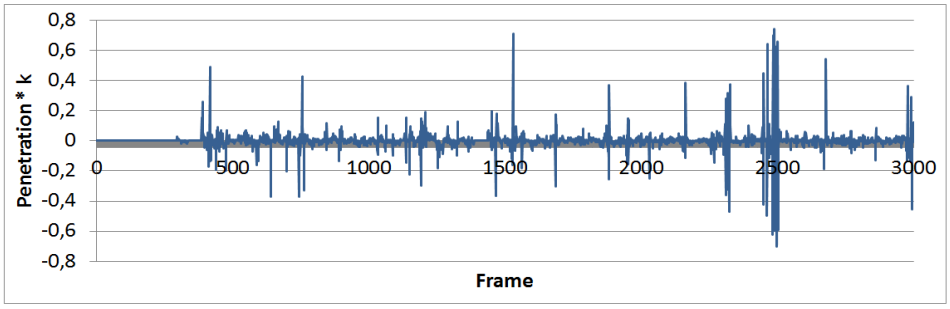


(c) Path3: Localised experiment

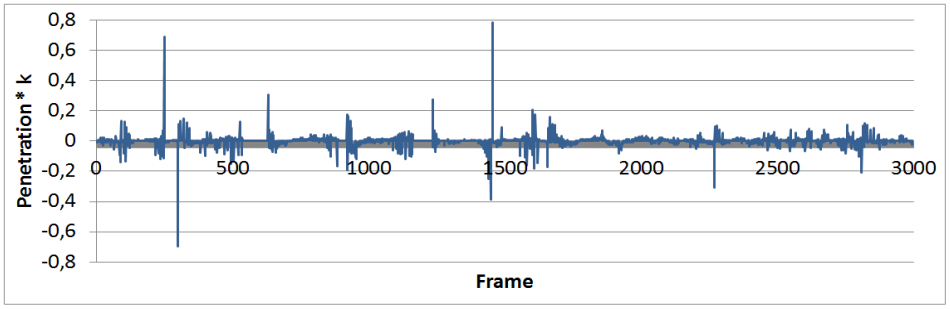
Figure 4.18: Penetration depth returned by the deformable collision handling method for Brain2, Brain3 and Brain 4 models responding to three different situations



(a) Path1: moving the tool around the surface



(b) Path2: Random freehand experiment



(c) Path3: Localised experiment

Figure 4.19: Variations in the force's modulus along different frames multiplied by the stiffness constant with a Brain3 model

account that the Phantom OMNI can return forces until 3.3N, our system has a comfortable margin not to saturate the haptic device. This is to some extent due to the chosen stiffness constant ($k = 0.3$) which softens the resulting force, and the low variations in the penetration depth which avoid high peaks in the force variation in consecutive frames.

As said before, our deformable collision handling algorithm uses a viscoelastic model to calculate the final force to be sent to the haptic device. Figure 4.20 is an example of the final viscoelastic forces calculated by our system with a Brain3 model. The experiment has been done by touching the brain model repeatedly in a concrete zone of the brain imitating the movements of a surgeon. The force variations in the three axis are also given afterwards.

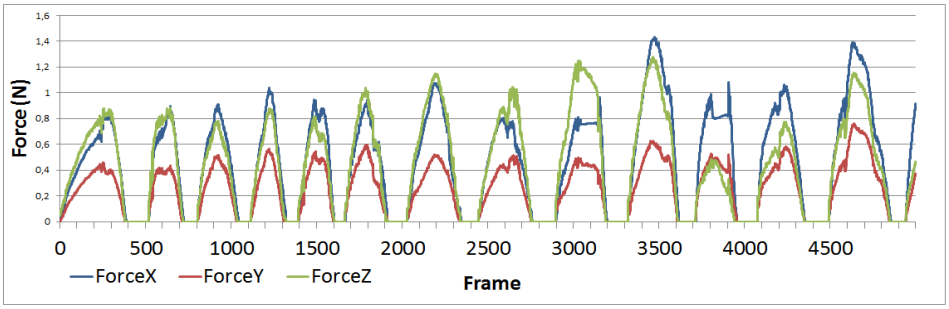


Figure 4.20: Viscoelastic forces returned by our collision handling algorithm with a Brain3 model

The maximum exertable force is never exceeded in this case and the subtle force variations in consecutive frames make the system a stable haptic environment. Moreover, with the aim of delimiting the experiments to the analysis of our collision handling method's behaviour, all the information has been collected omitting brain deformations. The forces given in the graphs are the result of interacting with brain models that behave as if they were rigid. In such a way, the behaviour of the algorithm can be examined without external parameters interfering. After all, a rigid body can be understood as a deformable body with infinite stiffness. In such a way, decreasing the model's stiffness and adding the mechanical properties of a real brain tissue, the realism of the contact felt by the user will always be improved. The mechanical properties of soft-tissues carry small deformations, so the forces returned to the haptic device are nearly the same.

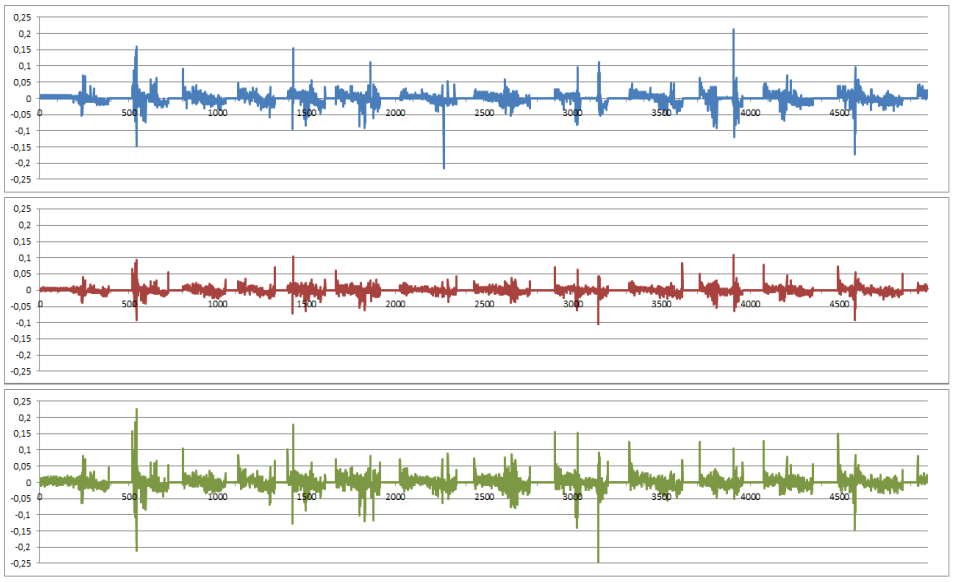


Figure 4.21: Variations in the viscoelastic force along different frames with a Brain3 model

4.5 Discussion

This chapter has described a deformable volumetric collision handling method which is integrated into a Brain Haptic Physical Simulator.

The collision handling includes an accurate collision detection algorithm that detects the interferences between a dynamic virtual tool and a deformable volumetric brain model represented by tetrahedra. The intersecting primitives (tetrahedra and points) are sent to the collision response module which computes the resulting force parameters (the contact normal and penetration depth). The collision detection process has been optimized in order to speed up the time consumption of determining the intersecting primitives.

The developed collision response and haptic control method computes the parameters needed to get the final force restitution based on the colliding tetrahedra and points. Finally, a viscoelastic force model calculates the force to be sent to the haptic device. According to the experimental results, our method gives an stable force restitution to the user.

Experiments have been performed for input models with different

tessellations. The results conclude that, with the exception of some concrete cases, our deformable collision detection and response method achieves the required minimum frequency with input models containing a big amount of tetrahedra. However, the performed improvement in the collision detection process minimizes these cases of the computational cost being higher than it should. Regarding to the force restitution, the returned force parameters and final result offer an stable contact with small force variations and rare force discontinuities. This makes our algorithm a suitable method to be used for training or rehearse.

Chapter 5

Optimal Voxel Size Computation

The content of this chapter has been published in:

> *Echegaray, G. and Borro, D. "A methodology for optimal voxel size computation in collision detection algorithms for virtual reality". Virtual Reality, Vol. 16, N. 3, pp. 205–213. September, 2012.*

5.1 Introduction

Real-time virtual reality applications require accuracy but are also time dependent, therefore in these environments the time consumption is particularly important. For that reason, when facing the problem of collision detection for a virtual reality application, we firstly focus our attention on optimizing time performance for collisions among objects. Spatial Partitioning algorithms have been broadly used in collision detection. In particular, voxel-based methods are simple and quick, but finding the optimum voxel size is not trivial. We propose a methodology to easily determine the optimal voxel size for collision detection algorithms. Using an algorithm which represents volumetric objects with tetrahedra as an example, a performance cost function is defined in order to analytically bound the voxel size that gives the best computation times. This is made by inferring and estimating all the parameters involved. Thus, the cost function is delimited to depend only on geometric data. By doing so, it is possible to determine the optimal voxelization for any algorithm and scenario. Several solutions have been researched and compared. Experimental results with theoretical and real 3D models have validated the methodology. The reliability of our research has also been compared to traditional experimental solutions given by previous works.

Even if the problem of detecting collisions seems purely mathematical, there are some issues which complicate matters considerably. For example, it is necessary to use the limited processing power as efficiently as possible in order to perform the queries within a concrete time frame. Moreover, points in space are represented by floats, which cause rounding errors that can result in a completely different environment behaviour. Additionally, the memory consumption of these detection methods must also be considered.

We have focused our attention on optimizing time performance for the computation of collisions. VR applications require accuracy but are also time dependent, therefore in these environments the time consumption is particularly important.

One of the common techniques used in collision detection is based on uniform spatial partitioning. These types of methods always have the problem of determining the voxel size which, being a user-defined parameter, can considerably change the performance of the method. If the number of voxels per axis is not optimized, the detection could result in execution times that are too long to use in real time. The best idea would be finding a method which automatically calculates the optimal size of the voxel for any scenario. Unfortunately, such a value depends on too many factors and finding an analytical method is a difficult task. That is why most of the works that use voxel-based algorithms have solutions inferred from experimental results. For instance, García-Alonso et al. (Garcia-Alonso et al., 1994) use OBBs subdivided into voxels, and have experimentally found that the optimal case is defining 512 voxels per object. Other research cases (McNeely et al., 1999) determine the discretization depending on the resolution they want to obtain. On the other hand, according to Gregory et al. (Gregory et al., 2000), the voxel size is set by multiplying the average edge length by a constant.

We propose a methodology (a sequence of steps) to calculate the optimal voxel size for algorithms based on Uniform Spatial Partitioning. The main contribution of this work is a sequence of steps to infer an analytical method that automatically determines the optimal voxel size based only on the geometry of the scene. This will conclude in an easy and rapid method for improving algorithm time-rates. A previous work (Borro et al., 2004) validated the followed method with objects represented by triangle meshes. Our proposal extends that work to other types of object representations (tetrahedral meshes) and offers a general methodology for any kind of collision detection algorithm based on voxels. Other research (Teschner et al., 2003) has determined that optimal performance is

achieved when a grid cell has the same size as the average edge length of all tetrahedra. We will compare our solution with this assertion.

Furthermore, we provide a set of experimental results obtained from various 3D models. The input models have been chosen varying geometry and complexity. They have been used to validate the methodology and compare our results with others proposed in the bibliography.

The remainder of the chapter is organized as follows. Section 5.1 gives some previous works on finding the optimal voxel size for the spatial subdivision method. Section 5.2 describes a collision detection algorithm based on tetrahedral meshes, which is later used to infer the formula and carry out experiments. Section 5.3 introduces the optimal voxel size problem and describes the proposed solution. In Section 5.4, experimental results are shown, followed by the conclusions in Section 5.5.

5.2 Description of the Collision Detection Algorithm

The collision detection algorithm we have used to infer the method is a particular case where objects are represented by tetrahedral meshes. In this approach, the collision problem is solved using a uniform spatial subdivision. This technique subdivides space into *cells*, which are introduced into a vector depending on their position in the world (Teschner et al., 2003). Using voxels, cell access is fast, since the calculation of the cell is made in constant time:

$$cell = floor\left(\frac{\mathbf{p} + \mathbf{world}}{\mathbf{cell\_size}}\right)$$

where **p** is a point, **world** comprises the size of the scenario in the three axes and **cell_size** represents the size of the voxels in each axis.

Once we have the indices for each cell, we just need to use a simple function to locate them in a vector:

$$ind = ind_k \times numCellsX \times numCellsY + ind_j \times numCellsX + ind_i$$

Most works based on voxels use hashing techniques in order to reduce memory consumption. The chosen hashing function can change the performance

of the method. Values given by this function should be uniformly distributed to achieve quality performance. The hash table size also influences, as the use of big tables reduces the possibility of placing several different cells to the same position. Therefore, the algorithm usually works faster. On the other hand, the memory consumption is higher. Some studies have been done on this topic (Teschner et al., 2003), which have found that if the hash table is significantly larger than the number of primitives, the risk of hash collisions is minimal. Besides, hash functions work most efficiently if the hash table size is a prime number (Cormen et al., 2001).

The diagram in Figure 5.1 describes the collision detection algorithm we have used.

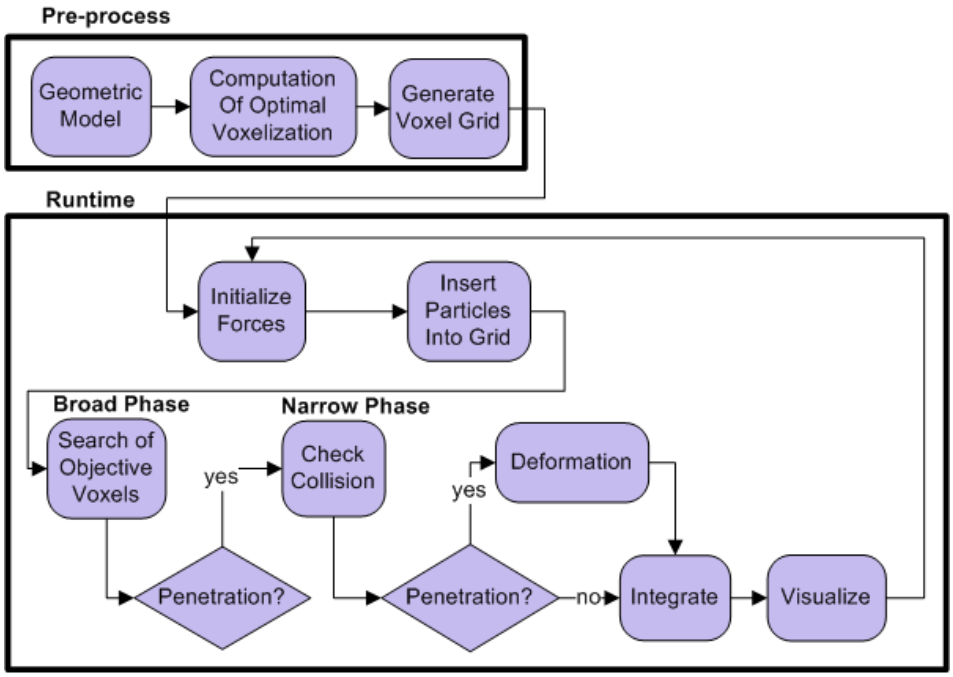


Figure 5.1: General design of the algorithm

In a pre-process block, the space is subdivided according to the chosen cell-size. Afterwards, at runtime, this grid is used to determine colliding voxels as well as checking primitives for intersection. These two steps are known as *broad phase* and *narrow phase*, respectively, according to the nomenclature of Hubbard (Hubbard, 1993). Once collisions have been detected, a collision response is

performed. This provides models with the convenient deformations which are calculated using a specific integration model.

The objects in the scene are represented by vertices (particles) and tetrahedra. From now on we will use *particle* and *vertex* without distinction. Once the voxel grid has been created, all the particles of a tetramesh are classified with respect to the cell in which they are placed. Afterwards, each cell has a list of references to particles contained in that cell. These lists will be modified in every single frame due to the mobility of the objects.

As mentioned before, the runtime block can be divided into two phases: the broad phase and the narrow phase.

In a first pass, all the particles in the scene are classified once again in the voxel grid. At the same time, the bounding boxes of the tetrahedra are recomputed based on their current deformed state.

In a second pass, the algorithm of (Teschner et al., 2003) is executed.

- **Broad Phase**: For each tetrahedra t of one input object, we get its bounding box $AABB_t$. Then, we identify the set voxels in the scene where $AABB_t$ lies. These voxels are called *objective voxels* (v_o). This delimits the whole problem to a smaller set of data.
  For each voxel in v_o, the particles belonging to the second object which lie inside it are analysed. Each one of them is checked for intersection with $AABB_t$. When a particle p is inside $AABB_t$ the second step (narrow phase) is performed.

- **Narrow Phase**: This step checks whether p intersects t or not. This is done using the barycentric coordinates of the vertex p with respect to the tetrahedron (Teschner et al., 2003).

The lists of the particles occurring in a cell need to be updated for every single object transformation. As has been shown before, using this structure there are a lot of primitives and particle pairs which can be rejected from intersection testing.

After detecting collisions between primitives, the collision response is performed. For this task, we have used the penalty method based on the penetration depth of intersecting objects. Even if there are many methods to estimate the penetration depth, the one used here computes a consistent penetration depth (Heidelberger et al., 2004b), which reduces collision response artifacts inherent to existing approaches. Moreover, a propagation scheme is

introduced (Heidelberger et al., 2004b) to approximate the penetration depth and direction for vertices with deep penetrations. This procedure will not be described in detail as the goal of this paper is optimizing the detection algorithm (next Section), not the response.

5.3 Analytical Computation of the Optimal Voxel Size

The idea of the Spatial Subdivision is to divide the space into convex regions called *cells*. These are determined by an identifier (calculated according to the coordinates of the cell), and are placed into a vector, in order to classify the interacting objects and determine relations between them.

The use of spatial partitioning techniques involves a basic initial problem: the 3D space is discretized with respect to a user-defined cell size, and the choice of that value can significantly change the efficiency of the algorithm. If the number of cells per axis is not optimal, the detection process may be too long to use in real time. Of course, this value changes depending on the scenario.

The optimal voxel size is the one which enables minimal calculation time cost in the collision detection algorithm. This will be delimited by these two events:

1. If cells are very large compared to the size of the model's tetrahedra, the number of primitives per cell increases, which involves a large amount of point-tetrahedron tests.
2. On the other hand, small cells cause a spreading of primitives to more cells. This will result in a lot of voxel-tetrahedron intersection tests.

What has been done is to extend a previous experience in this topic (Borro et al., 2004) to generalize a methodology valid not only for scenes of triangle meshes, but also for scenes with other types of object representations.

In the next section, the parameters which are supposed to affect the collision detection time in an instant are identified. Then, we will show how to refine this definition of parameters.

5.3.1 Methodology to Infer the Cost Function

This section describes the methodology to obtain an analytical solution for optimizing the voxelization.

The *cost function* of the algorithm is the mathematical expression that describes its behaviour, namely, it predicts the time it will take to solve a collision problem in a determined scenario.

The factors that could be involved a priori in the analytical formula are the following: the number of polygons in the scene, the average of their areas, the average length of the edges, and the average volume of the tetrahedra. Location of the objects and the contact volume could also affect the cost function of the algorithm.

If the cost function is constructed dependent of the voxel size, it is enough to find the value of this variable which makes the function minimum. What we present is an adaptation of the approach we used in a previous work (Borro et al., 2004) for the new algorithm presented in Section 5.2, proving that the methodology might be extended to other collision detection algorithms where objects are not represented by triangles. We will take the collision detection algorithm, infer the cost function of its behaviour and estimate all the parameters of the cost function using only geometrical data from the scenario.

As shown in a previous section, the collision detection algorithm has two levels of precision: voxels and tetrahedra. Each tetrahedron of the object goes through both levels in case of collision. So if t_o is the number of tetrahedra of one object, v_o (objective voxels) is the average number of voxels covered by the bounding box of a tetrahedron, and p_v is the average number of affected particles per voxel, the algorithm behaviour could be expressed as in Equation 5.1 (remember that the algorithm has three loops: one works through each tetrahedron of the object, another one through the objective voxels of each tetrahedron and the last one through the particles of each objective voxel):

$$z = t_o \cdot v_o \cdot p_v \tag{5.1}$$

This equation only considers the worst case: when one whole object is colliding with another. In a normal case, this does not happen, as the collision response algorithm does not let the objects penetrate each other to such a degree. A typical simulation does not have collisions, or at most a few tetrahedra intersect. That is why z should be separated into two cases:

$$z = t_o \cdot p_{nc} \cdot v_o + t_o \cdot (1 - p_{nc}) \cdot v_o \cdot p_v \tag{5.2}$$

where p_{nc} is the probability of no collision for each tetrahedron. Experiments made with different scenarios and objects have shown that this probability can

always be bounded between 0.98 and 0.99 when the physically based collision response algorithm is implemented (our experiments have used $p_{nc} = 0.99$). This means that the first term of 5.2 will have much more weight than the second one, as happens in a real simulation where only a few tetrahedra are in collision.

Even if the equation in 5.2 gives the same weight to the parameters v_o and p_v, this could be refined. The time it takes to determine if a voxel is empty of particles (t_v) is much lower than checking whether a particle collides with a tetrahedron or not (t_p). For that reason, we have defined a *ratio* r_t between particle and voxel times. It is defined as $r_t = t_v/t_p$. According to our experiments, checking particles for intersection is 5 times more expensive than the voxels time.

Apart from that, t_o is a constant value (it does not depend on the voxel size) and we do not need it to compute the minimum of z. So our z can finally be written as follows:

$$z = p_{nc} \cdot v_o + (1 - p_{nc}) \cdot v_o \cdot p_v \cdot r_t \tag{5.3}$$

Once the behaviour of the algorithm is established via the cost function, an expression of the parameters v_o and p_v needs to be found in order to associate them with the voxel size (the variable we want to solve) and the geometry of the scene (the only data we have).

5.3.2 Cost Function Parameters

This section presents an expression of the cost function which obtains a voxel size (δ) that minimizes the function value.

For a specific voxelization and a scene, v_o and p_v can be directly measured. Nevertheless, if we want to replace them in the expression of the cost function, they need to be approximated as a function of δ and the geometry data. Basically, we modify the derived expressions of (Borro et al., 2004) to work with tetrahedra, instead of triangle meshes.

We will deduce the expressions for both parameters in the next lines:

- v_o: As previously stipulated, this is the average number of voxels contained within the bounding box of a tetrahedron t, specifically the ones intersected by its AABB. So in order to express v_o, we first need to estimate the dimensions of the bounding box. In a simpler case, a way of constructing a box that always covers a triangle is by choosing the one in which the side is equal to the largest side of the triangle. So in our case, as the tetrahedra

can be in any space orientation, in the worst case the AABB involving a tetrahedron will be a cube in which the side, s_{AABB}, is the largest edge of the tetrahedron.

Once we have the dimensions of the AABB, and based on the voxel size, the number of voxels intersected by the bounding box in the worst case can be predicted.

$$v_o = \left\lfloor \left(\frac{s_{AABB}}{\delta} + 2 \right) \right\rfloor^3 \tag{5.4}$$

- p_v: The number of particles inside a voxel can be approximated finding the average number of triangles contained in the voxel. This could be done with different methods. We use the area of the tetrahedra facets (triangles) to approximate the maximum number of facets per voxel. As we use triangle areas, we can take the same approach of (Borro et al., 2004) to compute this parameter taking into account that the biggest triangle a voxel can contain is the one shown in Figure 5.2.

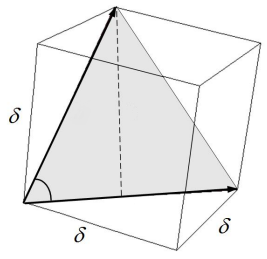


Figure 5.2: Largest triangle contained in a voxel

The area of that triangle (A_{vox}) is

$$A_{vox} = \frac{\delta^2 \sqrt{3}}{2} \tag{5.5}$$

Being A_t the real average area of the scene triangles, the number of facets per voxel (f_v) can be easily predicted:

$$f_v = \frac{A_{vox}}{A_t} = \frac{\delta^2 \sqrt{3}}{2A_t} \tag{5.6}$$

Once we estimate the number of triangles contained in a voxel and if each triangle joins three particles, the number of particles per voxel is:

$$p_v = 3 \cdot \frac{\delta^2 \sqrt{3}}{2A_t} \tag{5.7}$$

We already have all we need to get the optimal value of the cost function seen in Equation 5.3: v_o and p_v have been predicted in this section, and p_{nc} and r_t are constant numbers. Then, the cost function z only depends on the voxel size δ. Therefore, we will obtain the optimal voxel size by minimizing the function. This optimal value will be calculated in the pre-process stage and the time consumption for that work is obviously not significant. This can be done by solving the following equation:

$$\frac{dz}{d\delta} = 0 \tag{5.8}$$

5.4 Experimental results

Using a software that runs a set of simulations with different voxelizations, several experiments can be made in order to get their execution times and determine the optimal zone experimentally. That will be useful to validate our analytical solution. Approximating the times obtained in the set of simulations with a polynomial function $f(x)$ will allow us to experimentally determine the optimal value and the *optimal zone*. We understand as *optimal zone* the range of voxel sizes that give the smallest computation times for the algorithm: smaller or larger levels of voxelization than the ones in the optimal zone lead to greater frame rates.

The minimum of the polynomial function used for the approximation is considered the experimental optimal value. The optimal zone involves the values close to the optimal value, which have been delimited as the ones which do not raise the optimal value more than 1 millisecond.

We have performed two types of experiments: tests with simple and theoretical objects and tests using some standard 3D models. The primitive used in the simple tests is the sphere. Although in real simulations complex models are always going to be used, these first experiments can gather information about the method behaviour. Table 5.1 shows the number of tetrahedra of the simple models used to carry out the experiments.

The next lines list some criteria used throughout the entire experiment.

model	tetrahedra
Sphere01	41937
Sphere02	52326
Sphere03	60688
Sphere04	72631
Sphere05	80927

Table 5.1: Description of the models

- The developed software automatically runs an analysis for different voxel sizes. For each voxelization level, the same procedure is applied: two identical objects are placed into a position where they collide with each other. Then, necessary forces are applied to the objects (collision response, gravitational forces and so forth) and the simulation runs for 10 frames. Collision detection times for those 10 frames are used to calculate the average computation time.

- Analysed positions have not been chosen randomly. They have been selected in order to have an average number of 1% of the tetrahedra of the scene in collision. As the implemented collision response algorithm does not let the objects penetrate significantly into each other, positions in which more than the 1% of the tetrahedra are in collision will never happen. This value has been chosen after performing some experiments with real models in order to check the average number of colliding tetrahedra.

Figure 5.3 shows the results following the steps in the experiments. This one concretely belongs to a scene with two Sphere01 objects. It shows the average times of the experiments carried out with these two objects when using different voxel sizes. Although the cost function of the previous section depends on the voxel size (δ), in graphs and tables the number of voxels per axis (derived from δ) is used, considering that this value is easier to evaluate for the reader. For that reason, the given x values are the number of voxels per axis. The size of the scenario is $4x4x4$. The curve has been approximated with the polynomial function that best fits it. An *optimal zone* $[\delta_{min}, \delta_{max}]$ is defined by taking an interval around the *optimal value* δ_{opt}.

Each experiment $(\delta_{min}, \delta_{opt}, \delta_{max})$ is compared with two values: the first one is the voxel size guessed by our analytical formulation (δ_a). The result is also compared with one traditional solution (δ_{trad}) based on the average length of

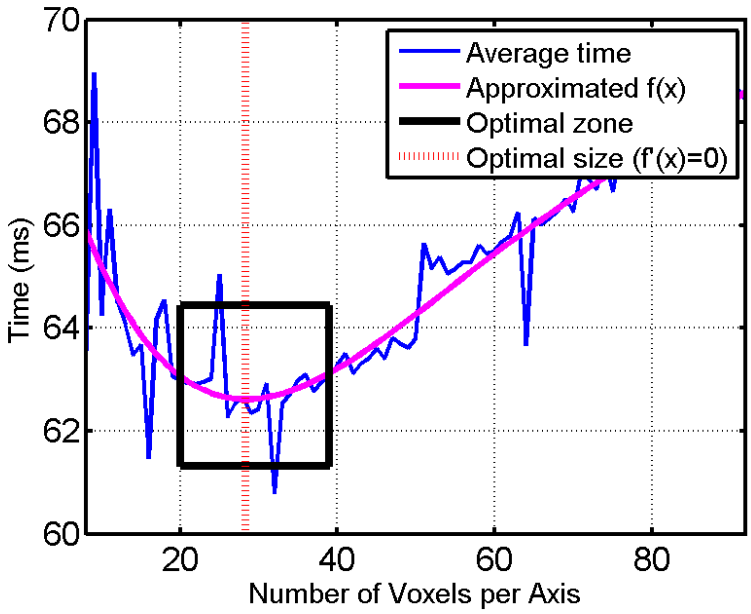


Figure 5.3: Experimental average times and approximation of the curve for two Sphere01 models colliding

the edges (Teschner et al., 2003). Table 5.2 gives the results of experiments with different objects.

	Sphere01	Sphere02	Sphere03	Sphere04	Sphere05
δ_{min}	20	22	16	8	20
δ_{opt}	28.33	28.16	34.04	26.63	37.87
δ_{max}	39	38	54	47	57
δ_{a}	28.73	30.94	32.49	34.48	35.65
δ_{trad}	41.90	45.16	47.40	50.29	51.90

Table 5.2: Experimental optimal, our analytical approach and traditional values for each pair of spheres

Note that the optimal value according to our equation lies inside the minimum and maximum experimental values. On the other hand, values given by other solutions differ considerably from the real experimental optimum. On some

occasions, they do not even lie within the optimal zone.

In order to make it easier to see the difference between experimental and traditional solutions as well as the analytical ones derived from our research, a relative error ε_δ has been used as in (Borro et al., 2004).

$$\varepsilon_\delta = \frac{|\delta_{opt} - \delta| \cdot \left(\left\lfloor \frac{\delta_{opt}}{\delta} \right\rfloor + \left\lfloor \frac{\delta}{\delta_{opt}} \right\rfloor \right)}{(\delta_{opt} - \delta_{min}) \cdot \left\lfloor \frac{\delta_{opt}}{\delta} \right\rfloor + (\delta_{max} - \delta_{opt}) \cdot \left\lfloor \frac{\delta}{\delta_{opt}} \right\rfloor} \tag{5.9}$$

If the error is zero, it means that δ matches the optimal size found experimentally (δ_{opt}). If $0 < \varepsilon_\delta \leq 1$, δ is not exactly the experimental optimum but it lies inside the optimal zone so we consider it as a good approximation.

model	ε_{δ_a}	$\varepsilon_{\delta_{trad}}$
Sphere01	***0.038***	1.272
Sphere02	***0.283***	1.727
Sphere03	***0.086***	***0.669***
Sphere04	***0.385***	1.162
Sphere05	***0.124***	***0.734***

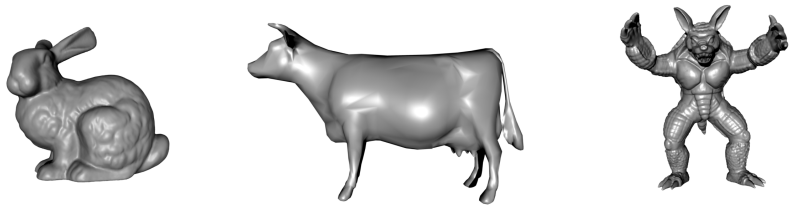
Table 5.3: Error ε_δ for both our analytical value (δ_a) and values given by traditional believes (δ_{trad})

Table 5.3 shows the relative errors for two cases: the first column is the error of the approximation made by our formula (based on the average area of the triangles that are part of the tetrahedra), respect to the optimal value obtained experimentally. The second column is the error for the voxel sizes used in traditional solutions (based on the average length of the edges), with respect to the optimal experimental value as well.

As it can be seen, our values are all inside the optimal zone ($0 < \varepsilon_{\delta_a} \leq 1$), very close to the experimental optimal voxelization levels. On the other hand, other solutions differ widely from the optimum, and in some cases are even outside the optimal zone. This could result in an important increase of the computation time. We have also made approximations based on other geometrical data like the average volume of the tetrahedra. Nevertheless we haven't included the results since the results based on areas are much more accurate.

Once experiments with theoretical models have been shown, results obtained for real models are given in the following figures and tables. Figure 5.4, Table

5.4 and Table 5.5 show the complexity of the real models, optimal values and computed ε_δ, respectively.



(a) Bunny (b) Cow (c) Armadillo

Figure 5.4: 3D Models used in the experiments: (a) Bunny (20462 tetrahedra), (b) Cow (50380 tetrahedra), (c) Armadillo (16377 tetrahedra)

	Bunny	Cow	Armadillo
δ_{min}	16	13	13
δ_{opt}	22.97	31.15	20.49
δ_{max}	34	54	37
δ_a	31.35	48.80	31.55
δ_{trad}	44.94	70.93	45.48

Table 5.4: Experimental optimal, our analytical approach and traditional values for each pair of 3D models

model	ε_{δ_a}	$\varepsilon_{\delta_{trad}}$
Bunny	***0.759***	1.992
Cow	***0.772***	1.741
Armadillo	***0.669***	1.516

Table 5.5: Error ε_δ for both our analytical value (δ_a) and values given by traditional methods (δ_{trad})

Once again, the optimal cell size calculated by our method lies inside the optimal zone obtained experimentally. Nonetheless, solutions given by traditional methods are close but do not always belong to the optimal zone. Even if these previous methods give an approximation of the optimal voxel size and may be valid for some types of applications, our new approach gets much closer to the exact optimal value.

5.5 Discussion

VR refers to computer-generated environments that can simulate physical conditions. One particularly important aspect that influences the realism of VR is the problem of real-time interactive collision detection. Time consumption is particularly important in such environments. For that reason, we have presented the concrete problem of optimizing collision detection algorithms based on Uniform Spatial Subdivision. This is made by determining the optimal voxel size. We have studied the difficulties of the approach and proposed a methodology to optimize the selection of the best voxel size. Thus, starting from a Spatial Partitioning collision detection algorithm, we have described the steps to follow, in order to easily estimate the most appropriate cell size to optimize the behaviour of the algorithm in any scenario.

The voxel size chosen in the past was usually selected experimentally. Some studies give an approximation of the best voxelization by carrying out experiments, which lead to a method for delimiting a suitable voxel size based on the geometry of the scene. These solutions however, came quite close to the optimum, but were not always as accurate as some applications demanded. Our proposal gives a general methodology to infer an analytical solution based on a cost function and approximates the optimal size satisfactorily. It has also been validated experimentally and compared to other approaches.

We have presented a particular case to validate the methodology with objects represented by tetrahedra. As said before, a previous work validated the method for triangle meshes, so with these two studies we involve most of the representations used in VR applications. Even if the particular solution presented here is dependent on a concrete algorithm, the cost function can be easily obtained from other algorithms in the same way so the same method can be applied to any algorithm.

Chapter 6

Prototypes' Description

Believing is half the cure
TOBA BETA

Part of this chapter has been published in:

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization". In XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012). San Sebastián, Spain. 2012.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data". In XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011), pp. 423–426. Cáceres, Spain. 2011.

6.1 Introduction

We have developed a Neurosurgery Simulator focused on patient-specific planning and surgical rehearsal of brain tumour resections. Due to the used patient specific data, the surgeon can practice complex procedures before actually performing the intervention, thus lowering the odds of human errors. Different areas need to be combined in order to construct a feasible and completely accurate tool. Therefore, the disciplines we have integrated are:

medical imaging, 3D geometrical reconstruction of tissues, real time volume rendering, physical modelling, simulation, collision handling, visualization and haptics. The development of our simulator is divided into three phases:

- Craniotomy Simulator: This first prototype simulates skull drilling using a haptic device that governs a virtual milling tool. The system provides the surgeon with the visual and force feedback correspondent to an actual craniotomy intervention.
- Brain Haptic Physical Simulator: This simulator operates on an accurate model of the brain tissue. The deformations of the brain model due to the impact of the surgical tool are emulated, giving a realistic idea of the physical behavior of the tissue.
- Integration: It consists of the combination of both the Craniotomy Simulator and the Brain Haptic Physical Simulator. It is constituted by a skull model obtained from real CT data and a model of the brain tissue inside it. The brain model is acquired from MRI data of the same patient. The surgeon can drill the skull and interact with the virtual brain with a haptic tool. He or she is advised whenever the milling tool touches the brain tissue while drilling the skull. Once the drilling task is accomplished, the tool is changed and the new task is enabled. The resulting simulator enables surgeons to practice actual operations, and will serve as a training tool for residents and students.

An additional fourth stage would involve the expansion of our simulator to a final Neurosurgery Simulator oriented to brain tumour resection. In addition to the features mentioned above, it would include the option of cutting the brain tissue in order to let the surgeon access the tumour and resect it.

The required realism has been reached with the development of three principal components: refined visual representation, haptic interaction and realistic physical behaviour including collision handling and physical simulation.

6.2 Craniotomy Simulator

Skull drilling is an essential task required in many surgical procedures. This section describes a volumetric bone drilling process, which involves the combination of several disciplines such as 3D reconstruction, physical behaviour

including collision detection and visual feedback, and an accurate haptic response. We have developed a real-time volumetric framework for a Craniotomy Simulator which uses patient specific data. The performed Craniotomy Simulator simulates skull drilling using a haptic device that governs a virtual milling tool. The system enhances the approximation of reality, when milling and drilling bone. It provides visual and force feedback correspondent to an actual craniotomy intervention.

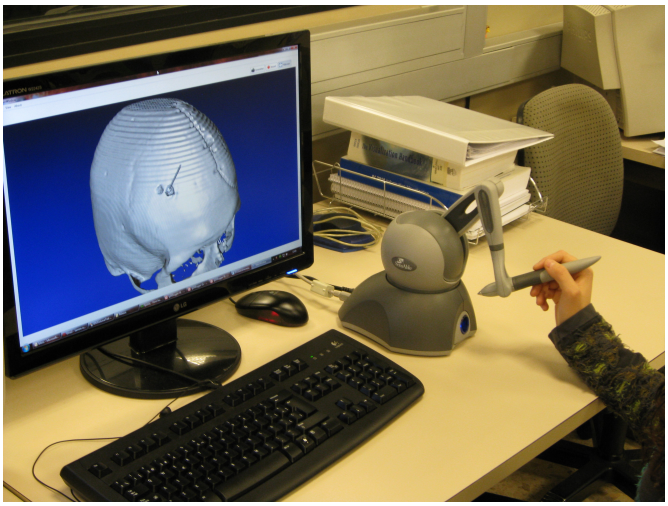


Figure 6.1: Craniotomy Simulator

The Craniotomy Simulator involves the design and development of the following areas: control, haptics, collisions and visualization. In order to cover these necessities, two threads have been designed and developed: on one hand, a main thread has been created which generates the simulation components and manages the visualization. On the other hand, a haptic thread is started from that first thread. This one manages the haptic device and its connection with the collision module, which handles interferences between the models. The haptic thread also gives the necessary force feedback to the user. Figure 6.2 gives a general idea of the interaction between different modules.

Each component will be briefly explained in the following lines. Although the principal aim of this thesis is the analysis, design and implementation of the collision handling, the rest of the modules also affect its functionality and effectiveness. For that reason, it is advisable to have a small idea of how each module behaves.

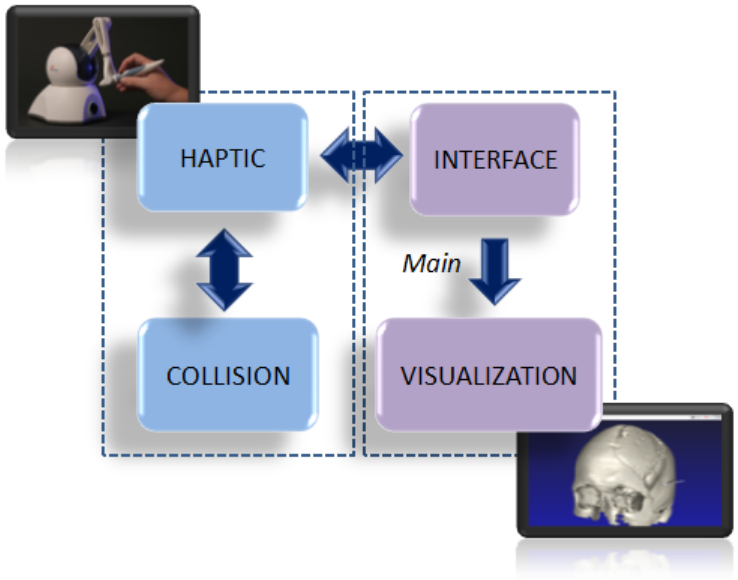


Figure 6.2: Craniotomy Simulator architecture

6.2.1 Main Thread

The Craniotomy Simulator is started in a main thread which creates all the components of the simulation. A simulator must start with the visualization of its components, and the module in charge of doing so is contained in this first thread. Hence, this thread transfers haptic position information to the visualization module with the aim of connecting the haptic thread with it. So the exchange of location information between the haptic module and the visual module is its concern. Apart from the position messages, control messages are also swapped as remote orders of starting and disconnecting.

Regarding to the graphical user interface (GUI), all the events for windows, mouse, keyboard or any user-interaction tools (except the haptic device) are handled here.

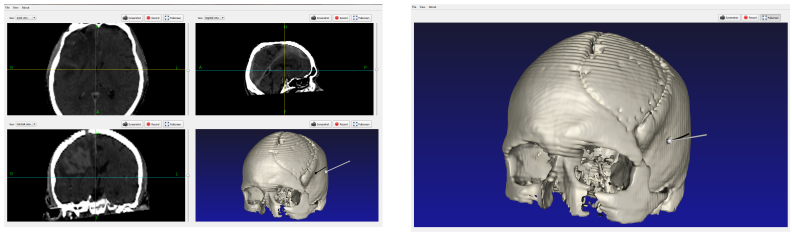
6.2.1.1 Visual Module

The main thread contains the visualization module and it maintains it updated about haptic position changes. The visual module must be informed about the

haptic movements as it is responsible for everything related to the graphic scene and must visually show the movements made by the user with the haptic device. It periodically receives position messages from the main thread, which have previously been calculated in the haptic thread.

The visual aspect and realism of the simulator greatly affects its teaching capability. Therefore, detailed medical images have been used to model the skull. In our case, CT data have been acquired since it renders the bone more clearly. Using patient-specific images increases the possibilities of the training as data from different patients with different anatomy and diseases can be used. This widens the variety of training scenarios and, consequently, increases the range of knowledge acquired through it. Due to this characteristic, the simulator can be used for rehearse in addition to training.

In Figure 6.3 the main window and an enlarged view of the skull and the tool of the Craniotomy Simulator can be seen. (a) shows the three orthogonal views of the skull and the 3D reconstruction of the environment. (b) gives the 3D reconstruction of the data. In such a way, it presents a user-friendly environment in which accurately drill the skull. Moreover, the user can move around with total liberty as in an actual intervention.



(a) (b)

Figure 6.3: (a) Craniotomy Simulator Interface and (b) an enlarged view of the skull

Tools and other objects are rendered using The Visualization Toolkit (VTK) (Will et al., 2006). It is an open source, cross-platform C++ library that supports a wide variety of advanced visualization and volume processing algorithms. We have implemented the visualization modules to be compatible with VTK. They are used to visualize the skull (from CT) in a fast and realistic way. Additionally, a renderer for the shadows of the tools is developed, which greatly increases the depth perception of the user.

Concretely, direct volume rendering is used to visualize the patient's real data. Raycasting is also used to visualize the skull. Nonetheless, the skull is completely opaque, so a direct isosurface rendering can be used, which is a more specific and optimized raycasting method. This method renders the isosurface defined by an isovalue, which is the density of the material to be visualized (the skull in this case). Samples where the densities are lesser than the given isovalue are completely discarded, and the renderer proceeds to check the next sample position. Samples with a value equal or bigger than the isovalue stop the ray traversal, and the illumination and shadowing are calculated as previously mentioned. In this rendering mode the opacity property is not given by the transfer function but is set to be completely opaque. These changes entail a drastic reduction on the number of calculations per ray, giving a huge performance boost.

6.2.2 Haptic Thread

The objective of the Haptic Thread is to manage the movement of the virtual tool around the environment and control the force restitution through the haptic device. For that issue, it uses the information obtained from the collision detection and response module. The haptic device we have used for this Craniotomy Simulator is a Phantom OMNI, a Sensable 6 DoF haptic device.

After initializing the proper data structures, the haptic thread reads the encoders of the haptic device and transforms the location and rotation information into the virtual environment's language. This information is then sent to both the visual module and the collision module. The first one needs the location data in order to reflect the movement of the virtual tool visually. Concerning to the collision module, it must be periodically informed about the tool's movements to adapt its data structures and detect possible restricted locations. Afterwards, the haptic thread asks the collision detection module to start its work and detect possible interferences between the skull model and the new position of the virtual tool. Once the collision response module has returned the collision information, even if no collision has happened, the haptic module applies the position changes known as "visual feedback" explained in Section 3.2.1. This must be done as the collision module sometimes changes the tool's position by a process we have called "sweep" (see Chapter 3). Finally, the haptic thread sends a three-dimensional force obtained from a viscoelastic force model to the haptic device so the user can feel the skull surface physically. This process is continuously running in a separate thread so it is absolutely independent from others.

6.2.2.1 Collision Module

The collision module is the responsible for detecting the intersections occurred in the virtual scene and calculating the necessary information to the be sent to the haptic thread. The design, implementation and analysis of the collision handling method for the Craniotomy Simulator has been one of the main objectives of this thesis. Hence, the collision detection method used in our Craniotomy Simulator is the one proposed in Chapter 3.

The proper haptic thread contains and manages the collision module. For that reason and in order to avoid instabilities, the collision module's frequency should not exceed 1000Hz rate. This means that the collisions calculation must be done in less than 1ms. If the time spent by the collision module to detect the interferences, determine the parameters that affect to the collision and calculate the restitution force is larger than 1ms, the haptic module must have alternatives to be able to give plausible force values. In our simulator, as mentioned in Chapter 3, when the haptic module does not receive any information from the collision module the force parameters received in the last frame are maintained. In this way the force stability is not altered.

6.3 Brain Haptic Physical Simulator

This section presents a real-time deformable simulation framework which uses patient specific data. The Brain Haptic Physical Simulator entails the design and implementation of areas such as 3D Reconstruction, haptic rendering and an accurate collision handling including visual feedback. Apart from that, due to the deformable nature of the brain model, a physical simulation must be added. The performed brain haptic simulator allows the user to interact with a brain model obtained from patient-specific MRI through a haptic device that governs a virtual surgery tool. All these aspects have been combined and organized as follows:

A main thread is responsible for creating all the elements of the simulator. One of the most important matters of this thread is the creation of the other three threads composing the simulator: The haptic, collision and the Simulation thread.

The haptic thread manages the movement of the virtual tool and controls the force restitution through the haptic device. It uses information obtained from the collision module.

The collision thread handles the interferences between the brain model and the

working tool. It is communicated with the haptic and Simulation thread by sending information about the forces needed to be applied to both the brain structure and the resecting tool when an interference has been found.

To end with, the Simulation thread computes the deformations to be applied to the brain mesh depending on the information received from the collision thread. Once the calculation is done, the visual module displays the new deformed state visually.

Figure 6.4 gives a general idea of the interaction between different modules.

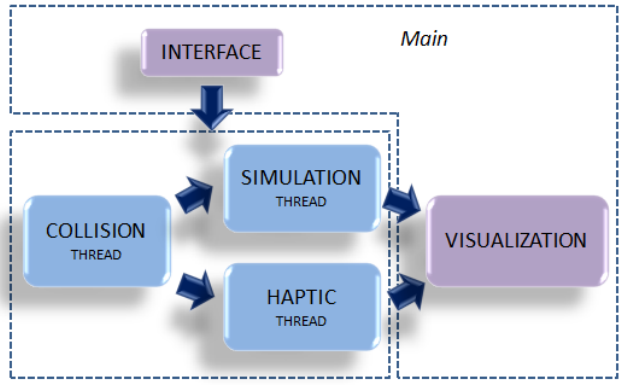


Figure 6.4: Brain Haptic Physical Simulator

6.3.1 Main Thread

Analogous to the Craniotomy Simulator, the main thread creates all the elements of the simulator: in this case the visual, collision and simulation modules together with the haptic device are initialized here. Once all the initializations have been made, the collision thread, the simulation thread and the haptic thread are ready to start so their start call is performed here.

As happened in the Craniotomy Simulator, the haptic and visual module interchange updated position and rotation information by means of this module. Apart from the location messages, this module also swaps control messages as remote orders of starting and disconnecting.

Concerning to the graphical user interface (GUI), all the events for windows, mouse, keyboard or any user-interaction tools are handled here.

6.3.1.1 Visual Module

The visual module is again the module in charge of managing everything related to the graphic scene. An interface module contained in the main thread periodically transmits position and rotation messages from the haptic thread to the visual module. In such a way, the visual module update the new state of the tool visually.

The deformations of the brain must also be reflected visually. When the user touches the brain by means of the virtual tool the simulation module calculates the subsequent deformation to be applied to the nodes of the brain. Such deformation must be visualized properly. Moreover, a visual feedback of the tool not penetrating the skull is also projected in this module. Hence, the user's perception of stiffness and robustness increases considerably.

An accurate visualization greatly affects the realism and validity of the simulator. Thus, our brain model is loaded from a real MRI image of the patient. Figure 6.5 shows an example of a brain model used in our Brain Haptic Physical Simulator. The use of patient-specific images gives the training capacity to our system, as data from different anatomies and diseases can be used.

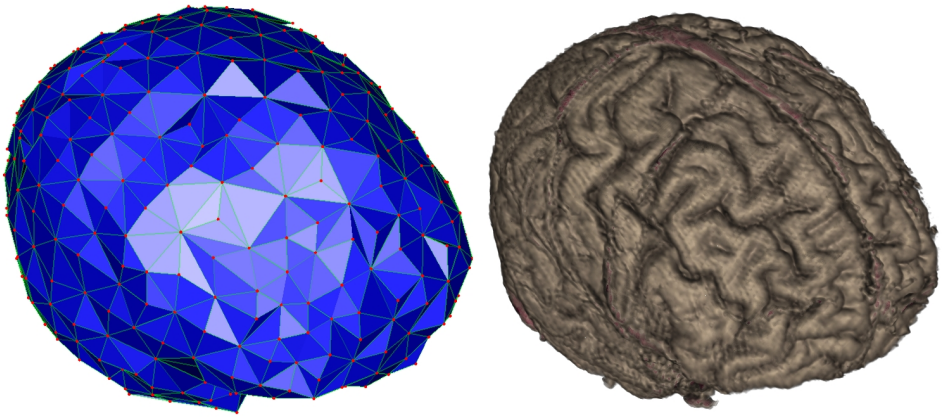


Figure 6.5: Tetrahedralized brain model

In order to achieve the stated accuracy the rendering of the skull is done using a volumetric isosurface rendering as explained in the visualization module of the Craniotomy Simulator. On the other hand we have the brain, which has to be rendered accurately with the calculated deformation. In this method, similar to the one presented in (Tejada and Ertl, 2005), a raycasting is performed with the tetrahedral data and the structured volume is used as the tetrahedrons' underlying

data. It should be noted that the tetrahedral mesh used in the raycasting is exactly the same used in the physical simulation. This allows us to render a correctly deformed brain using the patient MRI directly.

This approach achieves rendering speeds of 15-20 fps at 1680x1050 resolution, permitting a good interactive visualization.

An accurate and fast visualization facilitates the pre-operative tests to the surgeons and widens the knowledge acquired to students. In conclusion, this simulator can be used for rehearse in addition to training.

6.3.2 Haptic Thread

As stated in the Craniotomy Simulator, the objective of the haptic thread is again to handle the movement of the virtual tool around the environment. The control of the force restitution through the haptic device is also its concern. This is managed by using the information obtained from the collisions module and the visual feedback.

The haptic device used for the Brain Haptic Physical Simulator is obviously the same as the one used for the Craniotomy Simulator, since both simulators will be merged as a last stage. Thus, this simulator also makes use of a Phantom OMNI, a 6 DoF haptic device from Sensable.

The haptic thread first reflects the movement of the haptic device into the virtual tool by reading the device encoders of position and rotation. The virtual representation of the tool is assumed by a sphere surrounded by a point cloud (proxy).

Once the proxy is positioned at its new location, the haptic is ready to receive contact information from the collision module. The collision thread controls interferences between the scene objects continuously. In such a way, when the haptic thread is informed about the existing interferences, it can send the correct force feedback to the haptic device. In case of interference, the proxy position is updated by a god-object location method as in (Zilles and Salisbury, 1995b).

6.3.3 Collision Thread

Collision detection and response are vital aspects in a virtual surgery simulator, as they significantly affect the accurate real-time response and simulation

realism. The collision thread is responsible for keeping the environment free of interferences between the brain model and the resecting tool. The data-structures corresponding to the collision are initialized in the interface module from the main thread as a preprocess stage before creating the collision thread. Once created, the collision thread makes use of the collision module to run a collision step repeatedly. The collision module implements a detection algorithm based on a Uniform Spatial Subdivision method which has been detailed in Chapter 4.

Further, the contact information needed to give a haptic response to the interferences is also computed in this thread (Chapter 4). The haptic thread will periodically take the contact information and send it to the haptic device via force feedback.

Finally, the brain model also responds to the touch of the virtual tool by running a physical simulation framework. For that issue, it is necessary to compute a set of forces to be applied to the brain nodes as a response to the collision. These forces are also calculated in the collision thread.

6.3.4 Simulation Thread

Medical applications require both a realistic visualization and a realistic simulation of the behaviour of the tissue. The deformations of the brain must be driven by a realistic physics simulation.

Several approaches have been used in the past to drive physic based simulation. However, the use of FEM is the most relevant when accuracy is an important factor. These implementations are suitable for the simulation of objects that undergo small deformations and whose material behaviour can be considered linear. However the behaviour of soft-tissue is notoriously non-linear, and in many applications large deformations are also common. For this reason, non-linear FEM implementations are preferable in this work. The non-linearity is handled using a Total Lagrangian non-linear Explicit Dynamics (TLED) approach (Joldes et al., 2010).

The simulation is calculated in the GPU using CUDA in order to speed up the calculations. In such a way, for tetrahedral meshes of for instance 22k tetrahedra, real time is achieve by using a time step of 0.33ms. Figure 6.6 shows the deformation suffered by the texture of the brain when a virtual tool interacts with it.

The finite element method requires a discretized mesh for the material to be

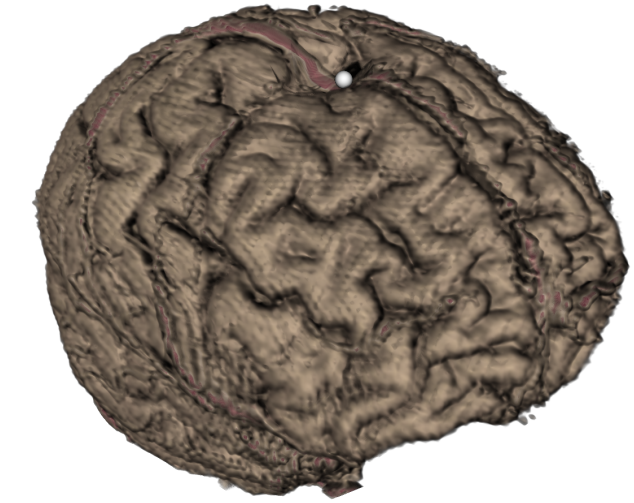


Figure 6.6: Deformation of the brain model when a virtual sphere intersects with it

simulated i.e. a tetrahedral or hexahedral mesh. This simulator uses a tetrahedral mesh extracted from the MRI data using the tool iso2mesh[1].

The soft-tissue parameters used in the simulation are not the ones related to the concrete patient. Nonetheless, the general mechanical properties of soft tissues are typically used for this kind of simulation. Table 6.1 gives the simulation parameters used in our Brain Haptic Physical Simulator to give a physical behavior to the brain model knowing that the scene objects are measured in millimetres.

parameter	value
Density ρ	0.000001
Young's E	30.0
Poisson ν	0.3
Gravity	9810

Table 6.1: Description of the simulation parameters

[1] http://iso2mesh.sourceforge.net/cgi-bin/index.cgi

6.4 Integration

The integration consists of the combination of both the Craniotomy Simulator and the Brain Haptic Physical Simulator explained in Sections 6.2 and 6.3. It is composed by a skull model obtained from real CT data which contains a brain tissue model inside it. The latter is obtained from MRI data from the same patient. By means of this simulator the surgeon first drills the skull using the haptic device that governs a virtual milling tool. If the tool touches the brain tissue in the drilling process warning messages are displayed in order to alert the user and avoid any damage to the patient. Once the drilling task has been accomplished, the tool changes and the user can interact with the virtual brain via the haptic device.

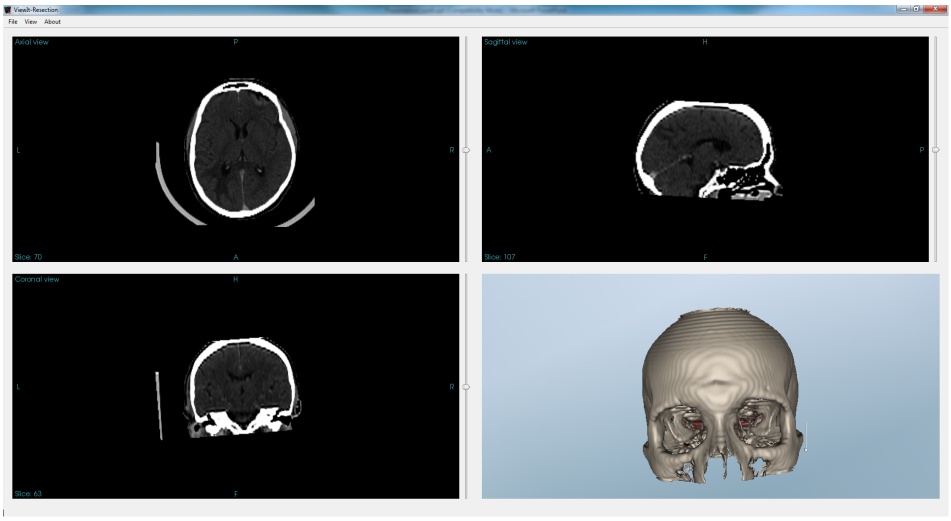


Figure 6.7: The main window of the integrated simulator

As explained in Section 6.2, the Craniotomy Simulator is composed of two threads: the main thread and the haptic thread. In the case of the Brain Haptic Physical Simulator detailed in Section 6.3, two additional threads are needed: the collision thread and the simulation thread. Thus, the integration of both simulators is constituted by four threads: the main, haptic, collision and simulation thread.

Both the drilling task and the interaction with the brain have a flag to inform whether one or the other is being accomplished. The algorithm runs different modules depending on the activated task. At the beginning of the operating process the craniotomy flag is active and the brain interaction is disabled. In such

a way, the algorithm performs the instructions correspondent to the Craniotomy Simulator. When the surgeon finishes the drilling task the craniotomy flag is disabled to let the brain interaction flag activate. This implies the start of the Brain Haptic Physical Simulator instructions. However, even if the interaction with the brain is disabled while performing the drilling task, the user wants to be advised if the drill touches the brain while drilling. For that reason, the collision thread corresponding to the Brain Haptic Physical Simulator is initialized and starts its run from the beginning of the drilling task despite its disabled flag. The start of each thread during the simulation process is illustrated in Figure 6.8.

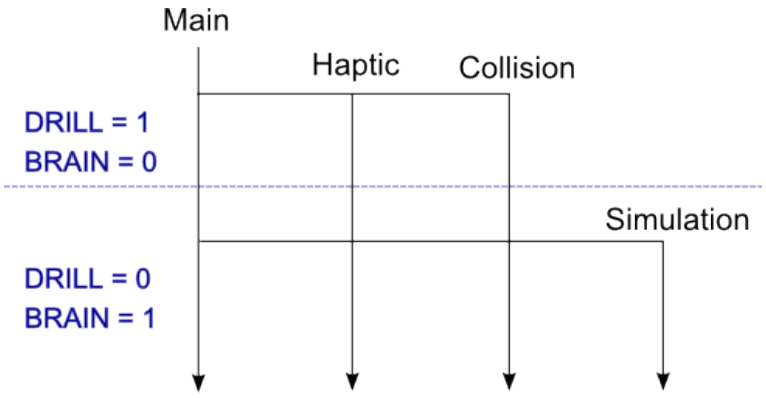


Figure 6.8: Flow of the threads during the simulation process

As well as in the two individual simulators, all the threads are created and initialized in the main thread. An interface and visual module comprise this thread. Apart from generating the components of the simulator, is also manages the connections between different threads and modules.

The interface module basically behaves as it did in the second stage of our simulator: the Brain Haptic Physical Simulator. The initializations of the visual, collision and simulation modules are made there, together with the startup of the haptic device. After creating and initializing all the modules, the collision thread, the simulation thread and the haptic thread are ready to be activated. The swap of starting and disconnecting control messages is also its concern. Moreover, the GUI is handled by the interface module by controlling all the events for windows, keyboard, mouse or any other user-interaction tools.

The integrated system visualizes a drilled skull at the time the brain is inside it (Figure 6.9). Thanks to the optimizations and specific rendering modules, interactive visualization rates between 15Hz and 20Hz are achieved at $1680x1050$

resolution. The reason behind such high resolution is the didactic nature of the simulator, in which a teacher can be showing the student the proper use of the simulator.

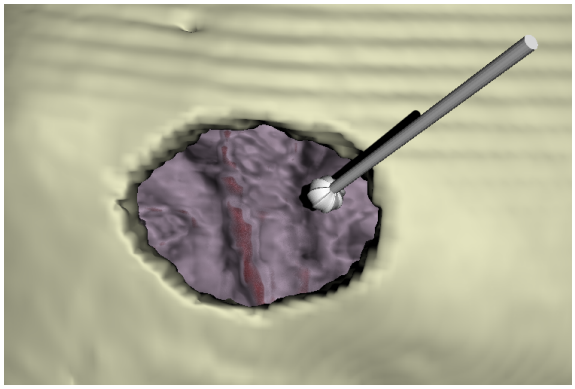


Figure 6.9: Drilled skull

One of the major changes added with respect to the Craniotomy and Brain Haptic Physical Simulators is the way in which the haptic thread works. The management of collisions changes depending on the volume the user is interacting with at each moment. The interaction with the skull (rigid body) and the brain (soft body) differs, so each case is treated in a different manner.

Remembering Figures 3.8 and 4.6, the first step in both cases is the reflection of the haptic device's movement into the virtual tool. This is made by reading position and rotation encoders. In the case of craniotomy, an additional sweep process is performed before reflecting the new position into the tool.

The second stage to be carried out is the obtaining of the collision information. The main difference between both flows in this case is the way in which they get the interference information. The first simulator just calls a function from the collision module which returns whether a collision is happening or not. It also returns the force parameters (penetration depth and contact normal). In the second case a collision thread is running outside the haptic thread, so the only thing to do is to take the necessary information from there.

The haptic interface of the Craniotomy Simulator gives a visual feedback even if no collision is found. On the contrary, in the Brain Haptic Physical Simulator a visual feedback is not needed when the objects do not interfere. Apart from that, the returned visual feedback differs for both simulators. So the integration solves this aspect separately.

To finalize, the last stage calculates and sends the correspondent forces to the haptic device. Although this step might seem the same for both simulators, the chosen force model is different for each one. We have incorporated a viscosity component to the Brain Simulator in order to achieve a higher realism. For that reason, when sending forces to the haptic device the object's material is also taken into account.

In conclusion, the integration process of the haptic thread requires a differentiation of the two different tasks. After moving the tool (this is made equally in both cases) the algorithm takes the first route if the craniotomy flag is active, and the second one if it is not.

Regarding to the collision thread, it only deals with the collision handling respective to the brain interaction. The interaction with the skull is managed in a collision module which does not work independently as a thread but it is called from the control loop of the haptic thread. At the beginning of the simulation both the collision module for rigid bodies and the collision thread for deformable objects are active. However, once the drilling task finishes the rigid collision detection does not work anymore.

Although the collision thread for deformable objects starts running from the inception of the simulation, the collision response is not available until the drilling task ends. During the craniotomy the only force feedback the user receives is the one related to the interaction with the skull. However, at the same time the collision module is detecting interferences between the tool and the skull, the collision thread detects collisions between the milling tool and the brain in order to advise the user whether the brain is being damaged while drilling. Therefore the collision detection is performed but the collision response is not applied neither to the haptic device nor to the brain model.

Figure 6.10 describes the connections between the haptic thread and the collisions in both the drilling process and the brain interaction task. In the first stage the haptic thread receives collision information about the rigid body interferences and gives a response to them. Apart from that, the collision thread detects interferences between the deformable body and the tool, but the haptic module does not receive any information about it. A force response is not applied for such interferences. When the drilling process ends, the rigid collision module stops and the information the haptic receives is the one related to the brain model. The response now corresponds to the brain model.

The deformable nature of the brain makes it necessary to simulate its

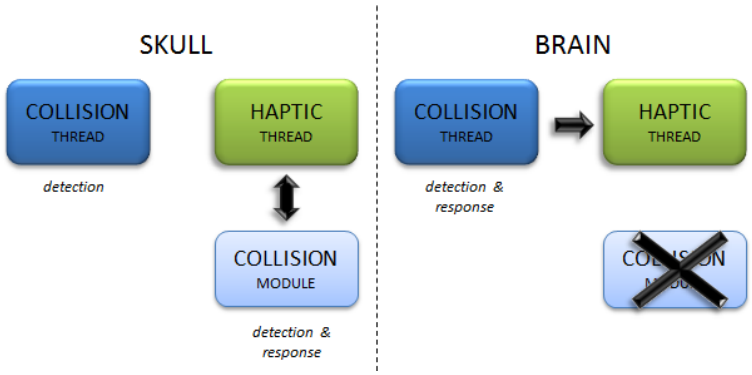


Figure 6.10: Interaction between the haptic thread and the collisions while drilling the skull and interacting with the brain

behaviour when the operating tool interacts with it. The simulation module is the one in charge of assigning a realistic physical behaviour to the brain model. Several approaches have been used in the past to drive physic based simulation. However, given the kind of materials present in biological models, specially the mechanical properties of soft tissues, and the amount of deformation they can suffer during a procedure, the previously mentioned non-linear FEM is the approach capable of providing the most realistic behavior. It is also the most relevant when accuracy is an important factor.

The method, material properties and tools employed in the integrated simulator for the simulation of the soft body are exactly the same used in the Brain Haptic Physical Simulator. The skull model does not need a simulation process due to its rigid nature, so the only simulation method used in the integration is the one relative to the second stage.

The resultant Neurosurgery Simulator (see Figures 6.11 and 6.12), created from the combination of the Craniotomy Simulator and the Brain Haptic Physical Simulator consists of a skull model which covers a brain tissue model from the same patient. The entire simulator can be a useful tool to train students or residents in real skull drilling procedures. Furthermore, the interaction of the virtual tool with the brain tissue enhances the immersion of the students into the intervention. In conclusion, this framework could avoid or at least reduce the time spent by students and residents attending long real interventions with the aim of being instructed in real neurosurgery tasks.

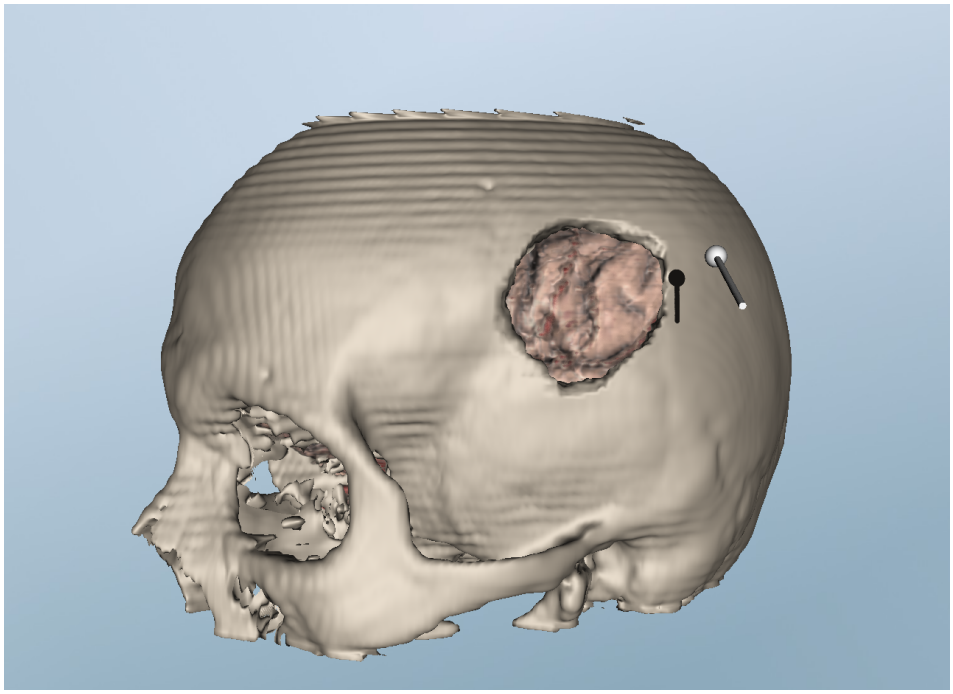


Figure 6.11: The multimodal Neurosurgery Simulator where the drilling task is being accomplished

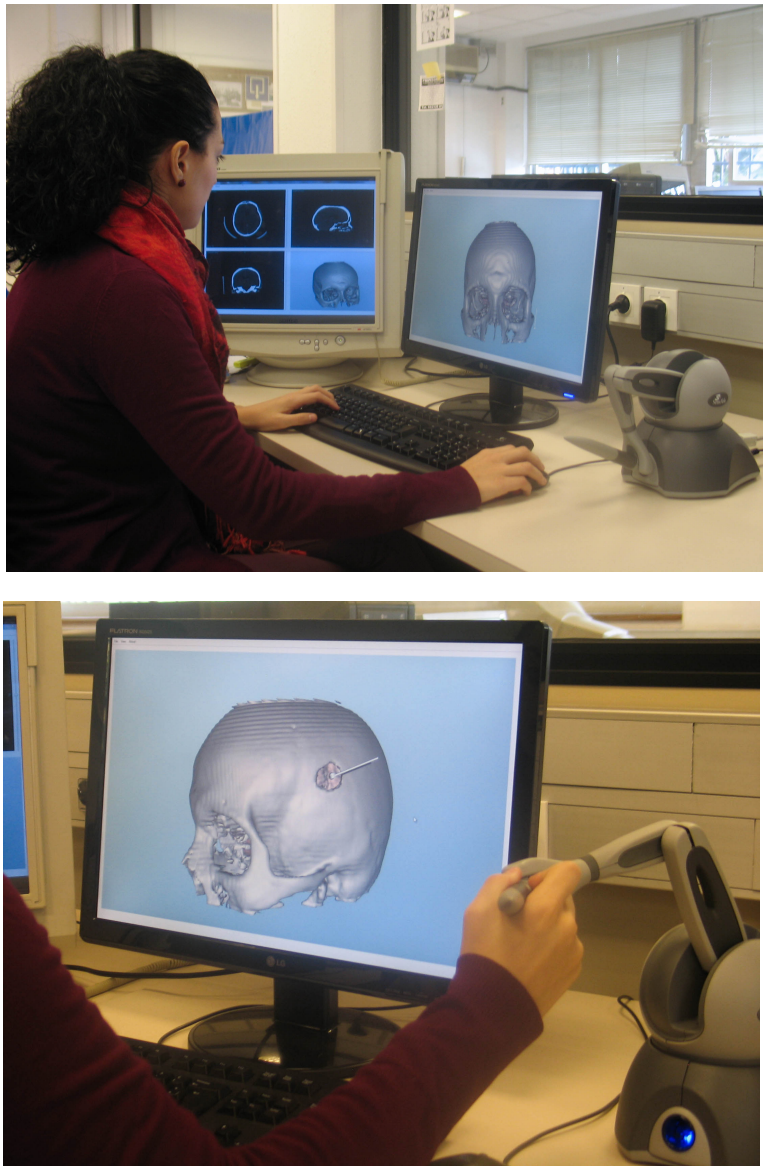


Figure 6.12: Integrated Neurosurgery Simulator framework

Part III

Conclusions and Future Work

Chapter 7

Conclusions and Future Work

7.1 Conclusions

We have presented a Neurosurgery Simulator with particular emphasis on the collision detection and response handling methods.

The Neurosurgery Simulator is divided into three phases. First, we implemented a Craniotomy Simulator that is based on real patient CT data and simulates skull drilling using a haptic device that governs a virtual milling tool. Thanks to the collision detection method developed for rigid volumetric data and the implemented collision response technique, the system enhances the approximation of reality when milling and drilling bone. It provides the surgeon with the visual and force feedback that corresponds to an actual craniotomy intervention.

The collision handling method implemented for this stage calculates a stable response in complex volumetric environments with discontinuities. While other simulators convert their volumetric data into hybrid or adapted structures with surface information, we maintain the original voxel structure and respond to collisions by solving adversities. Our method offers a realistic and pleasing force restitution without the need of transforming the original data into an adapted structure.

As a second stage, we have developed a Brain Haptic Physical Simulator. The user interacts with a brain model obtained from patient-specific MRI data through a haptic device that governs a virtual surgery tool. As a result of the deformable collision detection and response technique, the user can interact with the brain tissue and feel the forces via the haptic device. At the same time, the

corresponding deformations are reflected in the brain model.

In this case, the collision detection and response is performed for deformable objects that are represented by tetrahedra. This is based on the uniform spatial partitioning technique, which uses a predefined voxel size that can cause performance disorders if it is not chosen correctly. Thus, we have also presented a methodology for describing the steps to follow in order to easily estimate the most appropriate voxel size and optimize the behaviour of the algorithm. We later applied this methodology to get the most fitting voxel size for our environment and thus optimize the time consumption. The resulting stable and pleasing contact handling increases the feeling of interacting with real brain tissue.

The last phase involved the integration of both simulators into a multimodal Neurosurgery Simulator, as well as the incorporation of new components and features such as notifications or the ability to switch tools during the operation. The Neurosurgery Simulator offers the possibility of exploring the environment by interacting with both the skull and the brain's deformable tissue. While drilling the skull, warning messages alert the user if the tool is somehow touching the brain and thus causing any damage to the patient. Once the drilling task is finished, the working tool changes and the user can start the interaction with the brain.

In order to improve the realism of the simulation and increase the learner's immersion into the environment, accurate visual feedback was incorporated into all the simulators. This enhances the user's immersion and presence in the environments since the virtual tool is not allowed to penetrate the skull nor the brain visually.

All stages of this simulator can be a promising tool for training students in real neurosurgery procedures. The time students and residents spend attending long real interventions in order to be instructed in such tasks could be considerably reduced; it could even become unnecessary.

Additionally, it can be utilized as a suitable surgery rehearsal tool. Because the simulator uses patient-specific data, the surgery can be planned and practiced prior to the actual intervention. The results give a precise sensation of performing a real bone drilling intervention, and offer a useful tool according to surgeons from the "Clínica Universitaria de Navarra" (CUN).

A realistic haptic simulator describing a faithful behaviour of the tissues offers convincing benefits in comparison to other simulation tools:

- The precision and realism of the interaction support surgical anatomy

teaching and training for students, residents and specialists.

- The use of patient-specific data makes the simulator a plausible tool for rehearsal in addition to training. The surgeons can practice procedures before facing complex interventions.
- Accurate model simulation and an accurate environment can help for surgical equipment design considering dimensions, working angles and shapes.
- Moreover, the accessibility and suitability of medical instrumentation can also be analysed with this type of simulators.

7.2 Future research lines

As an extension of the work presented in this thesis, several possible research lines are still open. Some of them are:

- The first prototype of our Neurosurgery Simulator is a Craniotomy Simulator that enables skull drilling procedures. The collision handling algorithm for this simulator was implemented based on the volumetric representation of the skull model. Even though the solution ended up being as accurate and realistic as was required, it could be compared with other existing rigid collision detection algorithms for skull drilling procedures. As most of the simulators convert their volumetric data to adapted structures and handle collisions with this second representation, a comparison in terms of time-consumption or user perception could be interesting. However, our simulator behaves correctly with the patient data we employ. For that reason, the comparison would be interesting as an informational study for simulators where the base data are heavier and the computational cost causes problems.
- The second prototype proposed here is the Brain Haptic Physical Simulator. This was designed in order to allow the user to freely interact with the brain tissue of the patient. Nevertheless, the representation of the brain is assumed by a single tetrahedral mesh with the same physical properties for all its tetrahedra. In the real world a brain contains vessels, blood and different tissues. It also has its own pulse. All these components should be distinguished and properly simulated. This would involve additional

collision detection, response, and haptic feedback methods, as well as liquid simulation techniques. Additionally, having the information of real brain tissue properties and being able to simulate them would also imply a huge improvement of the simulator's realism.

- The integrated Neurosurgery Simulator enables the combination of both tasks: skull drilling and brain haptic interaction. Nonetheless, a complete Neurosurgery Simulator should add the possibility of manipulating the brain tissue by making cuts or absorbing serum with special tools. These are common tasks a neurosurgeon carries out during a neurosurgery operation, so the learning and training skills of students and residents would increase significantly with all these additional features. Therefore, an additional fourth stage of the simulator could extend it by including for example a volume cutting feature.

- The validity of our simulators should be proven through experimentation with expert individuals as the possible users are students or residents. The patient specific data make it possible to rehearse a specific procedure prior to the real intervention. Thus, validation experiments with experienced surgeons would be interesting in order to prove its rehearsal capacity. The simulator's reliability could be determined by simply asking experienced surgeons to compare it with real neurosurgery interventions. It could also be completed with analytical experiments of accuracy and stability.

- A voxel-size optimization methodology has also been proposed in this thesis. This methodology was later applied in order to optimize the performance of the Brain Haptic Physical Simulator and the final Neurosurgery Simulator. The study is valid for rigid and deformable bodies that are represented by triangle or tetrahedral meshes. Nevertheless, it could be interesting to generalize it in order to optimize the time performance for volumetric objects represented by densities. In this way, the Craniotomy Simulator could also be optimized.

Part IV

References

Appendix A

Generated Publications

Journals

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Physical simulation and visualization for a surgery simulator of skull drilling and brain interaction tasks". *IEEE Transactions on Visualization and Computer Graphics*, 2012. Waiting for acceptance.

Echegaray, G. and Borro, D. "A methodology for optimal voxel size computation in collision detection algorithms for virtual reality". *Virtual Reality*, Vol. 16, N. 3, pp. 205–213. September, 2012.

Conferences

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization". In *XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012)*. San Sebastián, Spain. 2012.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data". In *XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011)*, pp. 423–426. Cáceres, Spain. 2011.

Echegaray, G., Herrera, I., Sánchez, E., and Borro, D. "Design of a neurosurgery simulator for tumour resection". In *Medicine meets Virtual Reality - Italy: Applications of Virtual Reality to Medicine and Surgery*. Aula Magna dellaScuola Superiore Sant'Anna, Pisa, Italy. 2010.

References

Acosta, E. and Liu, A. "Real-time volumetric haptic and visual burrhole simulation". In *Virtual Reality Conference*, pp. 247–250. Charlotte, NC. 2007.

Acosta, E., Liu, A., Armonda, R., Fiorill, M., Haluck, R., Lake, C., Muniz, G., and Bowyer, M. "Burrhole simulation for an intracranial hematoma simulator". In *Medicine Meets Virtual Reality 15*, volume 125 of *Studies in Health Technology and Informatics*, pp. 1–6. I O S Press, Amsterdam. 2007. ISI Document Delivery No.: BLN86 Times Cited: 1 Cited Reference Count: 9 English Proceedings Paper.

Agus, M., Giachetti, A., Gobbetti, E., Zanetti, G., and Zorcolo, A. "Real-time haptic and visual simulation of bone dissection". *Presence: Teleoperators and Virtual Environments*, Vol. 12, N. 1, pp. 110–122. 2003.

Anderson, T. C. L. "Using virtual reality technology for learning design skills". In *ED-MEDIA*, p. 752. Boston, USA. 1996.

Ar, S., Montag, G., and Tal, A. "Deferred, self-organizing bsp trees". 2002.

Asenjo, O. N. *Multipoint collision detection algorithm for bone milling and drilling in a haptic vr surgery simulator*. PhD thesis, Royal Institute of Technology. 2008.

Bharath, C. "Medical simulation in em training and beyond". volume 18(1), pp. 18–19. Mount Sinai Hospital. 2006.

Biesel, D. and Gross, M. "Interactive simulation of surgical cuts". In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, PG '00, pp. 116–125. Washington, DC, USA. 2000.

Borro, D., García-Alonso, A., and Matey, L. "Approximation of optimal voxel size for collision detection in maintainability simulations within massive virtual environments". *Computer Graphics Forum*, Vol. 13-24, 2004.

Brown, J., Montgomery, K., claude Latombe, J., and Stephanides, M. "A microsurgery simulation system". In *In Medical Image Computing and Computer-Assisted Interventions (MICCAI)*, pp. 137–144. 2001.

Carvalho, J. A. "Virtual reality and ophthalmology". *Revista Brasileira De Oftalmologia*, Vol. 71, N. 1, pp. 40–47. 2012. ISI Document Delivery No.: 909TL Cited Reference Count: 25 Portuguese Article 0034-7280.

Colgate, J. E., Stanley, M. C., and Brown, J. M. "Issues in the haptic display of tool use". In *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, p. 140. 1995.

Cooper, J. B. and Taqueti, V. "A brief history of the development of mannequin simulators for clinical education and training". *Quality and Safety in Health Care*, Vol. 13, N. 1, pp. 11–18. 2004.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms* (ISBN: 0-262-03293-7, 9780262032933). MIT Press, Cambridge, MA, USA, 2nd edition. 2001.

Denson J.S., A. S. "A computer-controlled patient simulator". *JAMA: The Journal of the American Medical Association*, Vol. 208, N. 3, p. 504. 1969. 10.1001/jama.1969.03160030078009 0098-7484 doi: 10.1001/jama.1969.03160030078009.

Echegaray, G. and Borro, D. "A methodology for optimal voxel size computation in collision detection algorithms for virtual reality". *Virtual Reality*, Vol. 16, N. 3, pp. 205–213. September, 2012.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization". In *XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012)*. San Sebastián, Spain. 2012.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Physical simulation and visualization for a surgery simulator of skull drilling and brain interaction tasks". *IEEE Transactions on Visualization and Computer Graphics*, 2012. Waiting for acceptance.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data". In *XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011)*, pp. 423–426. Cáceres, Spain. 2011.

Echegaray, G., Herrera, I., Sánchez, E., and Borro, D. "Design of a neurosurgery simulator for tumour resection". In *Medicine meets Virtual Reality - Italy: Applications of Virtual Reality to Medicine and Surgery*. Aula Magna dellaScuola Superiore Sant'Anna, Pisa, Italy. 2010.

Fuchs, H., Kedem, Z. M., and Naylor, B. F. "On visible surface generation by a priori tree structures". In *SIGGRAPH '80 Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pp. 124–133. New York. 1980.

Garcia, G. and Corre, J. F. L. "A new collision detection algorithm using octree models". In *Intelligent Robots and Systems '89. The Autonomous Mobile Robots and Its Applications. IROS '89. Proceedings., IEEE/RSJ International Workshop on*, p. 93. 1989.

Garcia-Alonso, A., Serrano, N., and Flaquer, J. "Solving the collision detection problem". *IEEE Comput. Graph. Appl.*, Vol. 14, pp. 36–43. May, 1994.

Giorgi, C., Pluchino, F., Luzzara, M., Ongania, E., and Casolino, D. S. "A computer assisted toolholder to guide surgery in stereotactic space". *Acta neurochirurgica. Supplement*, Vol. 61, pp. 43–45. 1994.

Gissler, M., Dornhege, C., Nebel, B., and Teschner, M. "Deformable proximity queries and their application in mobile manipulation planning". In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, ISVC '09, pp. 79–88. Berlin, Heidelberg. 2009.

Gissler, M., Schmedding, R., and Teschner, M. "Time-critical collision handling for deformable modeling". *Computer Animation and Virtual Worlds*, Vol. 20, N. 2-3, pp. 355–364(10). June, 2009.

Gleason, P., Kikinis, R., Altobelli, D., Wells, W., III, E. A., Black, P. M., and Jolesz, F. "Video registration virtual reality for nonlinkage stereotactic surgery". *Stereotactic and Functional Neurosurgery*, Vol. 63, pp. 139–143. 1994.

Govindaraju, N. K., Knott, D., Jain, N., Kabul, I., Tamstorf, R., Gayle, R., Lin, M. C., and Manocha, D. "Interactive collision detection between deformable models using chromatic decomposition". *ACM Trans. Graph*, Vol. 24, pp. 991–999. 2005.

Gregory, A., Lin, M. C., Gottschalk, S., and Taylor, R. "Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback device". *Computational Geometry: Theory and Applications*, Vol. 15, pp. 69–89. February, 2000.

He, X. and Chen, Y. "Bone drilling simulation based on six degree-of-freedom haptic rendering". In *EuroHaptics*, pp. 147–152. 2006.

Heidelberger, B., Teschner, M., and Gross, M. "Real-time volumetric intersections of deforming objects". In *VMV'03*, pp. 461–468. November, 2003.

Heidelberger, B., Teschner, M., and Gross, M. "Detection of collisions and self-collisions using image-space techniques". In *JOURNAL OF WSCG*, pp. 145–152. 2004.

Heidelberger, B., Teschner, M., Keiser, R., Müller, M., and Gross, M. "Consistent penetration depth estimation for deformable collision response". In *VMV'04*. 2004.

Hubbard, P. "Interactive collision detection". In *IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, pp. 24–31. oct., 1993.

Joldes, G. R., Wittek, A., and Miller, K. "Real-time nonlinear finite element computations on gpu - application to neurosurgical simulation". *Computer Methods in Applied Mechanics and Engineering*, Vol. In Press, 2010.

Kockro, R. and Hwang, P. "Virtual temporal bone: An interactive 3-dimensional learning aid for cranial base surgery". *Neurosurgery*, Vol. 64, N. 5(Suppl 2), pp. 216–229. 2009.

Kockro, R. A., Stadie, A., Schwandt, E., Reisch, R., Charalampaki, C., Ng, I., Yeo, T. T., Hwang, P., Serra, L., and Perneczky, A. "A collaborative virtual reality environment for neurosurgical planning and training". *Neurosurgery*, Vol. 61, N. 5, pp. 379–391 10.1227/01.neu.0000303997.12645.26. 2007.

Larsson, T. and Akenine-Möller, T. "Strategies for bounding volume hierarchy updates for ray tracing of deformable models". Technical report, Centre, Maelardalen University. 2003.

Lee, H. P., Audette, M., Joldes, G. R., and Enquobahrie, A. "Neurosurgery simulation using non-linear finite element modeling and haptic interaction". In *Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, And Modeling*, volume 8316 of *Proceedings of SPIE*. Spie-Int Soc Optical Engineering, Bellingham. 2012. ISI Document Delivery No.: BAP67 Times Cited: 0 Cited Reference Count: 21 English Proceedings Paper 83160H.

Lee, J. Y., Mucksavage, P., Kerbl, D. C., Huynh, V. B., Etafy, M., and M.McDougall, E. "Validation study of a virtual reality robotic simulator-role as an assessment tool". *Journal Of Urology*, Vol. 187, N. 3, pp. 998–1002. 2012. ISI Document Delivery No.: 892CY Times Cited: 0 Cited Reference Count: 16 English Article 0022-5347.

Lemole, M. G., Banerjee, P. P., Luciano, C., Neckrysh, S., and Charbel, F. T. "Virtual reality in neurosurgical education: Part-task ventriculostomy simulation with dynamic visual and haptic feedback". *Neurosurgery*, Vol. 61, N. 1, pp. 142–149. 2007.

Lin, M. C. and Manocha, D. "Collision and proximity queries". 2003.

Lombardo, J.-C., paule Cani, M., and Neyret, F. "Real-time collision detection for virtual surgery". In *In Computer Animation*, pp. 26–28. 1999.

Lorensen, W. E. and Cline, H. E. "Marching cubes: A high resolution 3d surface construction algorithm". In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pp. 163–169. New York, NY, USA. 1987.

Luciano, C., Banerjee, P., Florea, L., and Dawe, G. "Design of the immersivetouch: a high-performance haptic augmented virtual reality system". In *11th International Conference on Human-Computer Interaction*. Las Vegas, NV. July, 2005.

Madera, F. A., Laycock, S. D., and Day, A. M. "Detecting self-collisions using a hybrid bounding volume algorithm". In *Proceedings of the 2010 Third International Conference on Advances in Computer-Human Interactions*, ACHI '10, pp. 107–112. Washington, DC, USA. 2010.

Magnenat-Thalmann, N. "Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity". *Comput. Graph. Forum*, pp. 155–166. 1994.

Magnus, G. E. *Haptic and visual simulation of a material cutting process: A study focused on bone surgery and the use of simulators for education and training*. PhD thesis. 2006.

Mantovani, F. "VR Learning: Potential and Challenges for the Use of 3D Environments in Education and Training". 2001.

McNeely, W. A., Puterbaugh, K. D., and Troy, J. J. "Six degree-of-freedom haptic rendering using voxel sampling". In *26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pp. 401–408. New York, NY, USA. 1999.

Morris, D., Girod, S., Barbagli, F., and Salisbury, K. "An interactive simulation environment for craniofacial surgical procedures". In KG, W. J. H. R. H. H. M. G. P. R. R. R. V., editor, *MMVR (Medicine Meets Virtual Reality)*, volume 111, pp. 334–341. 2005.

Morris, D., Sewell, C., Barbagli, F., Salisbury, K., Blevins, N. H., and Girod, S. "Visuohaptic simulation of bone surgery for training and evaluation". *IEEE Computer Graphics and Applications*, Vol. 26, N. 6, pp. 48–57. 2006.

Pan, J. J., Chang, J., Yang, X. S., Zhang, J. J., Qureshi, T., Howell, R., and Hickish, T. "Graphic and haptic simulation system for virtual laparoscopic rectum surgery". *International Journal Of Medical Robotics And Computer Assisted Surgery*, Vol. 7, N. 3, pp. 304–317. 2011. ISI Document Delivery No.: 820PZ Times Cited: 1 Cited Reference Count: 37 English Article 1478-5951.

Payandeh, S. and Shi, F. H. "Interactive multi-modal suturing". *Virtual Reality*, Vol. 14, N. 4, pp. 241–253. 2010. ISI Document Delivery No.: 838HX Times Cited: 0 Cited Reference Count: 41 English Article 1359-4338.

Petersik, A., Pflesser, B., Tiede, U., Höhne, K., and Leuwer, R. "Realistic haptic volume interaction for petrous bone surgery simulation". *CARS*, 2002.

Pflesser, B., Petersik, A., Tiede, U., Höhne, K., and Leuwer, R. "Volume cutting for virtual petrous bone surgery". *Computer Aided Surgery*, Vol. 7, pp. 74–83. 2002.

Renz, M., Preusche, C., Pötke, M., Kriegel, H.-P., and Hirzinger, G. "Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm". In *Eurohaptics*. 2001.

Ruspini, D. C., Kolarov, K., and Khatib, O. "The haptic display of complex graphical environments". In *Computer Graphics*. 1997.

Salisbury, K., Conti, F., and Barbagli, F. "Haptic rendering: introductory concepts". *Computer Graphics and Applications, IEEE*, Vol. 24, N. 2, p. 24. 2004. 0272-1716.

Satava, R. M. "Virtual reality surgical simulator". *Surgical Endoscopy*, Vol. 7, N. 3, p. 203. 1993. 0930-2794.

Schuemie, M. J., van der Straaten, P., Krijn, M., and van der Mast, C. A. "Research on presence in virtual reality: A survey". *CyberPsychology and Behavior*, Vol. 4, N. 2, pp. 183–201. 2001.

Smith, A., Kitamura, Y., Takemura, H., and Kishino, F. "A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion". In *IEEE Virtual Reality Annual International Symposium*, VRAIS '95, pp. 136–. California, USA. 1995.

Tejada, E. and Ertl, T. "Large steps in gpu-based deformable bodies simulation". *SIMULATION MODELLING PRACTICE AND THEORY*, Vol. 13, N. 8, pp. 703–715. NOV, 2005.

Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D., and Gross, M. "Optimized spatial hashing for collision detection of deformable objects". In *VMV (Vision, Modeling and Visualization)*, pp. 47–54. Munich, Germany. 2003.

Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., and Volino, P. "Collision detection for deformable objects". In *EUROGRAPHICS*, volume 23, pp. 119–139. 2004.

Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., and Volino, P. "Collision detection for deformable objects". In *EUROGRAPHICS*, volume 23, pp. 119–139. 2004.

van den Bergen, G. "Efficient collision detection of complex deformable models using aabb trees". *Journal of Graphics Tools*, Vol. 2, N. 4, pp. 1–13. 1998.

Walsh, C. M., Sherlock, M. E., Ling, S. C., and Carnahan, H. "Virtual reality simulation training for health professions trainees in gastrointestinal endoscopy". *Cochrane Database Of Systematic Reviews*, N. 6, 2012. ISI Document Delivery No.: 957VD Times Cited: 0 Cited Reference Count: 66 English Review 1469-493X CD008237.

Wang, P., Becker, A. A., Jones, I. A., Glover, A. T., Benford, S. D., Greenhalgh, C. M., and Vloeberghs, M. "A virtual reality surgery simulation of cutting and retraction in neurosurgery with force-feedback". *Comput. Methods Prog. Biomed.*, Vol. 84, N. 1, pp. 11–18. 2006. 1647856.

Wang, P., Becker, A. A., Jones, I. A., Glover, A. T., Benford, S. D., Greenhalgh, C. M., and Vloeberghs, M. "Virtual reality simulation of surgery with haptic feedback based on the boundary element method". *Comput. Struct.*, Vol. 85, N. 7-8, pp. 331–339. 2007. 1231584.

Wang, Z., Tang, Z., Wang, T., Chen, M., Liu, D., and Tian, Z. "A virtual brain for stereotactic planning and supporting neurosurgery". *The International Journal of Virtual Reality*, Vol. 4, N. 4, 2000.

Wen, X., Qiang, Z., Weijia, K., and Enmin, S. "A simulation system of otology microsurgery based on distributed virtual reality technology". *Zhongguo yi liao qi xie za zhi = Chinese journal of medical instrumentation*, Vol. 34, N. 3, pp. 183–5. 2010. English Abstract; Journal Article Chinese 1671-7104.

Will, S., Ken, M., and Bill, L. *The visualization toolkit an object-oriented approach to 3d graphics, 4th edition* (ISBN: 1-930934-19-X). Kitware Inc. 2006.

Wong, G. K., Zhu, C. X., Ahuja, A. T., and Poon, W. S. "Craniotomy and clipping of intracranial aneurysm in a stereoscopic virtual reality environment". *Neurosurgery*, Vol. 61, N. 3, pp. 564–8; discussion 568–9. 2007.

Wong, W. S.-K. and Baciu, G. "Gpu-based intrinsic collision detection for deformable surfaces: Collision detection and deformable objects". *Comput. Animat. Virtual Worlds*, Vol. 16, N. 3-4, pp. 153–161. 2005. 1089874.

Zhang, H., Cao, X. H., Cui, G. Z., and Zhang, C. L. "Virtual reality system for spine surgery with haptic device". In *2011 International Conference On Energy And Environmental Science-Icees 2011*, volume 11 of *Energy Procedia*. Elsevier Science Bv, Amsterdam. 2011. ISI Document Delivery

No.: BYK25 Times Cited: 0 Cited Reference Count: 8 English Proceedings Paper.

Zilles, C. and Salisbury, J. "A constraint-based god-object method for haptic display". In Society, I. C., editor, *International Conference on Intelligent Robots and Systems*, volume 3. 1995.

Zilles, C. and Salisbury, J. "A constraint-based god-object method for haptic display". In Society, I. C., editor, *International Conference on Intelligent Robots and Systems*, volume 3. 1995.