UNIVERSITY OF NAVARRA

SCHOOL OF ENGINEERING

DONOSTIA-SAN SEBASTIÁN

Volumetric Visualization Techniques of Rigid and Deformable Models for Surgery Simulation

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

Imanol Herrera Asteasu

under the supervision of

Diego Borro Yágüez and
Carlos Buchart Izaguirre

October, 2013

delivered by
EG EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

Nire guraso eta arrebei.

Agradecimientos

Muchas son las personas que me han ayudado y acompañado durante estos años, y se me antoja difícil agradecer a todos ellos tal y como realmente se merecen. Son tantas las personas a las que agradecer, que es casi imposible que no me olvide de alguno, así que, para empezar: ¡Gracias a todos!

No hubiera podido empezar esta tesis, si no fuese por mis directores Diego y Carlos. Diego, por introducirme en el mundo de la investigación y enseñarme a moverme en él (y que algunos problemas no son mios si no de mi yo futuro). Carlos, por su parte, por adoptarme como su 'padawan' y dejarme algunas joyas de sabiduría como: *No saber algo también es saber*. A ambos, por vuestros consejos, ayuda y apoyo cuando los he necesitado. Sin vosotros esta tesis no hubiese sido posible: ¡Gracias!

Por supuesto, estos años hubiesen sido mucho más tediosos si no fuese por todos los compañeros de trabajo, con los que he compartido tanto tiempo y me han ayudado cuando lo he necesitado. Por supuesto he de empezar por agradecer a mi compañera de batallas, Goretti, que supo mantenerme optimista frente a los más extraños errores, alegrar los días más largos y, casi tan importante, seguirme cuando me ponía a cantar (Carabirubau). Pero no me olvido de agradecer a los demás: a Hugo, que fue quien me pego el 'carabirubi' y ha aguantado mis extrañas manías, Ibai que tanta compañía me ha hecho con sus chistes y comentarios extraños (tienes el puesto de ojito derecho abierto...). A Alba por esas conversaciones políticas y deportivas, a Aitor (¡Moar!) por quien repaso mis frases tres veces antes de decirlas. A Fran por las conversaciones frikis que hemos tenido (¡Dj Fostia!), a Ibon con el que siempre he tenido discusiones sumamente interesantes. A Alberto por los ánimos y consejos que me ha dado, a Iker por enseñarme todo sobre la F y a Josune por introducirme el mundo de la docencia. A todos, ¡gracias!

Podría escribir una tesis entera con todas las personas a las que agradecer la compañía y consejos que me han dado durante estos años: Gorka, Pedro, Borja, Valero, Luis, Denis, Ilaria, Sergio, Ainara, Álvaro, Xabi, Imanol P., Maider, Emilio, Leire, Ángel, Ana, Pablo, Ainhitze, Xabi y a otros tantos que seguro he olvidado incluir. ¡Gracias!

Una mención especial he de hacerle a Ainhoa, que se ha mantenido a mi lado durante todo este tiempo y más. Ha aguantado mis eternas explicaciones, ha sabido animarme siempre que lo he necesitado, ha sabido motivarme cuando yo ya estaba harto, en resumen, por estar siempre donde yo más la necesitaba. Sin ella, y sin su ánimo cuando mi motivación fallaba, no hubiese acabado esta tesis. ¡Gracias!

Tampoco puedo dejar de lado a mi kuadrila, que ha sabido aguantarme durante tantos y tantos años: Ane (que merece especial mención por su ayuda con ciertas fórmulas), Bagües, Itsaso, Azkarate, Aiora, Madina, Ibañez, Mikel, Zabaleku, Ander, Unai, Ruben, Zigor, Maitane, Aitor y Laffage.

No puedo acabar esta sección, sin agradecer a aquellos que durante tanto tiempo me han cuidado y educado. A mis padres, Manuel y Maria Luisa, les debo un agradecimiento especial por todo lo que han aguantado a mi lado y por haberme apoyado siempre en aquello que yo quería. Definitivamente, sin vuestro apoyo esta tesis no existiría, por lo que, ¡Gracias! También he de agradecer a mis hermanas, Maider y Maitane, por haberme apoyado durante estos años.

Por último, también me gustaría agradecer a la Universidad de Navarra tanto por permitirme cursar el doctorado con ellos como por la beca Universidad de Navarra que me concedieron, sin ellos esta tesis no hubiese sido posible. De la misma manera me gustaría agradecer a Alejo Avello y al Director del Departamento, por permitirme desarrollar esta tesis en el Departamento de Mecánica del CEIT.

Abstract

Virtual reality computer simulation is nowadays widely used in various fields, such as aviation, military or medicine. However, the current simulators do not completely fulfill the necessary requirements for some fields. For example, in medicine many requirements have to be met in order to allow a really meaningful simulation. However, most current medical simulators do not adequately meet them. One of these requirements is the visualization, which in the case of medicine has to deal with unusual data sets, i.e. volume datasets. Additionally, training simulation for medicine needs to calculate and visualize the physical deformations of tissue which adds an additional challenge to the visualization in these types of simulators.

In order to overcome these limitations, a prototype of a patient specific neurosurgery simulator has been developed. This simulator features a fully volumetric visualization of patient data, physical interaction with the models through the use of haptic devices and realistic physical simulation for the tissues. This thesis presents a study about the visualization methods necessary to achieve high quality visualization in such simulator.

The different possibilities for rigid volumetric visualization have been studied. As a result, improvements on the current volumetric visualization frameworks have been done. Additionally, the use of direct volumetric isosurfaces for certain cases has been studied. The resulting visualization scheme has been demonstrated by an intermediate craniotomy simulator.

Furthermore, the use of deformable volumetric models has been studied. The necessary algorithms for this type of visualization have been developed and the different rendering options have been experimentally studied. This study gives the necessary information to make informed decisions about the visualization in the neurosurgery simulator prototype.

Resumen

La simulación por ordenador con realidad virtual es ampliamente utilizada hoy en día en diversos campos tales como la aviación, la medicina o el entrenamiento militar. Sin embargo, los simuladores actualmente disponibles no son capaces de cumplir todas las necesidades en algunos de estos campos. Por ejemplo, en la simulación médica existen varias restricciones que han de cumplirse para proporcionar una simulación útil. La mayoría de simuladores médicos actuales no cumplen estos requerimientos, lo que reduce su utilidad. Uno de estos requisitos es la visualización, que requiere el uso de datos volumétricos además del cálculo y visualización de las deformaciones físicas, añadiendo un reto adicional.

Para poder superar estos retos, un prototipo de simulador de neurocirugía capaz de usar datos de pacientes ha sido desarrollado. Este simulador dispone de un visualizador totalmente volumétrico, una interfaz de interacción física a través de controladores hápticos y una simulación realística para los tejidos blandos. Esta tesis presenta un estudio sobre los métodos de visualización necesarios para dicho simulador.

Se han estudiado las diferentes opciones disponibles para visualización volumétrica rígida, desarrollando mejoras sobre las infraestructuras de visualización existentes. También se ha estudiado el uso de isosuperficies volumétricas en ciertos casos, pudiendo apreciarse el resultado en un prototipo intermedio de simulador de craneotomía.

También se han estudiado los métodos de visualización volumétrica deformable necesarios para un simulador de dichas características. Los algoritmos necesarios para esta visualización han sido desarrollados y se ha realizado un estudio experimental para indagar en los resultados de las diferentes opciones disponibles que permitirá una toma de decisiones justificada con datos empíricos.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Chapter 1

Introduction

Science is a way of thinking much more than it is a body of knowledge.
Carl Sagan

Through the last century, medicine has exponentially improved its understanding of the human body and its illnesses. At the same time, it has discovered many different ways to cure or prevent them. However, this increase of the knowledge has brought along an increase of the necessary knowledge, and thus, higher learning times to master the different possible fields in medicine.

It is widely known that practice is one of the best methods when it comes to teaching the necessary knowledge. Regrettably, practising in medicine is highly risky as the cost of an error can be tremendous. For this reason, simulators are particularly coveted in medicine as they allow to practice and learn a procedure without risking a patient's life.

1.1 Surgery simulation

In the early days of modern medicine, most learning and practice was done using cadavers. By performing procedures or dissecting them in front of an audience the techniques and details about the human body were taught, as seen in Figure 1.1.

Later on, this practice was reduced mostly to surgery practice for students in order to give them some experience before performing any

Figure 1.1: A painting by Rembrandt called *The Anatomy Lesson of Dr. Nicolaes Tulp*. It shows the dissection of a cadaver to explain the musculature of the arm in 1632.

real task. Nonetheless, the lack of other appropriate alternatives has made the practice with cadavers the standard teaching method up to this day. However, the scarcity of cadavers available and their cost encouraged the pursuit of a more available, more economic teaching method.

The advent of simulators in the 1930s with the invention of the Link Trainer, a pioneering flight simulator, showed the potential of this teaching method. As a result, many researchers found in simulation the perfect combination of cost, availability and accuracy needed for medical training. However, the technical limitations and the limited medical knowledge available at the time delayed the creation of medical simulators.

One of the first simulator presented, was called *Resusci-Anne* (Laerdal, Wappingers Falls, New York, USA). This simulator consist of a mannequin that can be used to train cardiopulmonary resuscitation (CPR). This mannequin was presented in 1960, and thanks to its simplicity and usability is still used nowadays with some minor modernizations.

However, the first computer controlled medical simulators presented was not presented until 1969, when the *Sim One* (JS and S, 1969) was made available. This simulator was a computer controlled mannequin capable of replicating physiologic responses such as dilating pupils, heartbeat or chest movement due to breathing. This allowed the students to gain more experience, and at an increased speed.

Sim One proved that the simulation of physiologic responses was not only desirable, but possible. However, due to the limited technology available at the time, the simulator's price was too high as it required much more computational capability than the one usually available.

The evolution of the computers led to a rapidly decreasing price as well as a exponentially increased computational power. Thanks to these advances, the computers needed nowadays for these calculations are commonplace, making the spread of simulators a reality.

1.2 Evolution of virtual reality

It is evident that simulators offer a high variety of solutions for learning, and that they present an enormous potential for medicine. However, the physical construction of the needed devices is a costly operation.

Virtual reality simulators are the natural evolution of the aforementioned simulators. They can be used to simulate virtually any known procedure, be it a simple wound stitching to a complex neurosurgical procedure, with a relatively low cost. However, this has not been always possible, as the computers available a mere decade were not capable of performing this complex calculations efficiently. As the computational power has grown, so have the simulators and the fidelity of its models.

One of the components with a major impact in this increase is the graphic processing unit (GPU). The GPU is responsible for the visual fidelity a simulator can offer, and their improvement has enabled the evolution of graphic fidelity, and with it, the evolution of virtual reality simulators.

In the last decade, the computational power of the GPUs have increased at an accelerated rate as it can be observed in Figure 1.2. Thanks to this, nearly photo realistic visualizations can be achieved in real time greatly enhancing the immersion of the user in the virtual world, increasing the teaching capabilities with it.

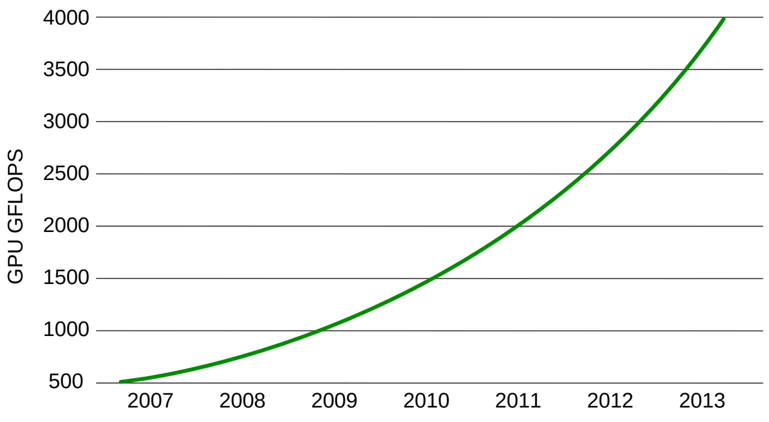


Figure 1.2: Chart illustrating the increase of the computational power of NVIDIA GPUs.

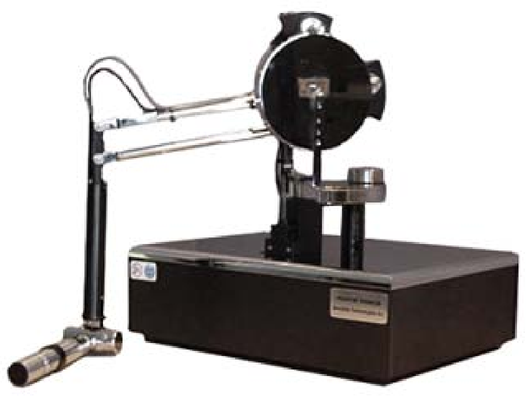


Figure 1.3: A 6 degree of freedom (DoF) haptic device, the Sensable Phantom Premium.

Additionally, the proliferation and evolution of force feedback capable input devices (haptic devices) added a missing information vector to virtual reality. Thanks to the use of haptic devices, such as the one shown in Figure 1.3, the user can feel the virtual world response to its interaction, in contrast to just seeing it. This exponentially increases the teaching capabilities of virtual reality, as it enables them the teaching of motor skills.

Thanks to this exponential improvement of virtual reality and its visual accuracy a whole new group of medical simulators appeared, virtual reality medical simulators. These have the advantage that do not require a physical copy of the object to simulate, decreasing their costs. And, thank to the offered realism and accuracy, are the most rapidly evolving simulators nowadays.

1.3 Virtual reality surgery simulator overview

In order to compete in terms of quality and teaching prowess a virtual reality simulator must have physical interaction, physical accuracy and visual fidelity. As it is shown in Figure 1.4, these features require the combination of different technologies and algorithms.

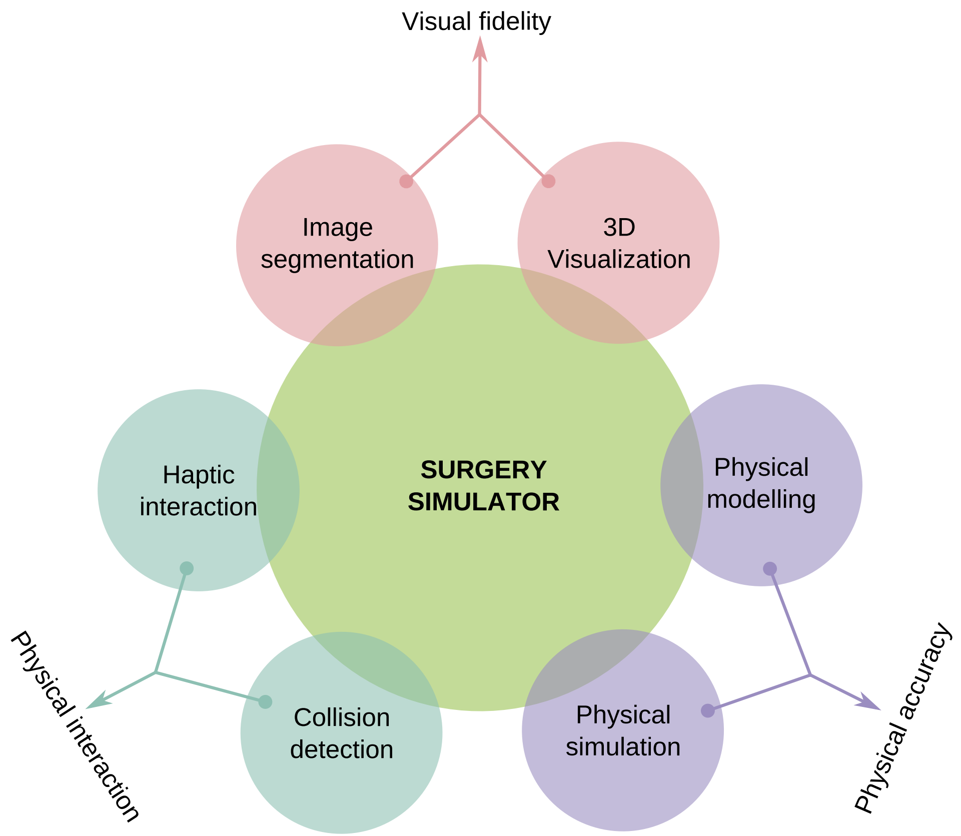


Figure 1.4: Basic diagram of a virtual reality surgery simulator.

The physical interaction will allow the user to interact with the virtual world by manipulating objects as well as feeling them. To do so, a fast and accurate collision detection algorithm is needed to realistically convey the interactions between the virtual tool and the rest of the virtual world. Additionally, a correct haptic interaction must be developed, so that the user can feel these interactions.

These interactions must be coupled with a realistic physical behaviour to increase the immersion of the user and present them with the correct reactions to their actions. This will enable showing the user the reactions of the world to the performed interaction. Additionally, the physical simulation will feed the forces back to the haptic device, increasing the accuracy. For this purpose, first the properties of the materials and objects to be simulated must be known. With this information, and the input from the haptic, the physical simulation will calculate the corresponding result.

But, in order to make the most of the simulator, the user must be able to see the results of the interactions beside feeling them. For this reason, the visual fidelity becomes a major necessity. In order to achieve the highest possible visual fidelity, using patients' real data is desirable, in order to account for the small individual differences. Although that is not always possible, the use of at least one medical dataset is mandatory. This volumetric dataset must be segmented to select which parts to be visualized, and with which rendering mode. Once this segmentation is done, these data have to be rendered with highest quality possible, be it using the usual polygonal approach (by extracting surfaces from the data) or directly visualizing the data with volumetric rendering.

1.4 Motivation

Simulation is widely used for training and rehearsing difficult or unusual actions in several fields. Moreover, the use of simulators has become mandatory in many fields such as aviation. However, the simulators available for some disciplines do not completely fulfil the requirements of reliability and accuracy needed. For example, due to the lack of adequate alternatives, every year neurosurgery students must attend real interventions carried out by expert neurosurgeons in order to learn specific procedures. The primary goal of medical training simulators is to transfer to the trainees the skills and experience needed to perform surgical procedures before practising on patients.

Two primary factors have to be taken into account to achieve these goals: presence and immersion. Presence is the feeling of being part of the virtual world by interacting and feeling the virtual objects by means of a haptic device, which boosts the learning of motor skills. On the other hand, immersion is the feeling of being in the virtual world, not just through interaction but also through hearing and sight. Immersion can be increased through the use of bigger and more immersive displays such as head mounted displays, and by improving the realism of the virtual world which covers the graphical appearance as well as the physical behaviour. It has been proven in various experiments (Bétrancourt and Tversky, 2000; Goldstone and Son, 2005) that the visual realism of the virtual world significantly improves the knowledge transferring capability of a simulator.

Neurosurgery simulators require of a high degree of precision and realism, which have not been achieved in simulators to this date. However, as medicine is an ever-changing and continuously updating field, simulators are being actively researched and many promising advancements have been done over the last few years (Thomas, 2013). The main sources of difficulties are the haptic interaction, the realistic visualization of the medical data and the physical simulation of the tissues.

In order to overcome these difficulties, the goal of this work is to develop a visualization module for a neurosurgery simulator focused on patient-specific training of brain tumour resection surgery. The use of patient specific data in the simulator, coupled with a high quality rendering, will provide a very precise and realistic visualization. To achieve this realism different areas will have to be combined: medical imaging, visualization, real time volume rendering, physical modelling and simulation, collision handling and haptic interaction.

The high visual realism and simulation accuracy will improve the experience gained by the users with the simulator, offering clear advantages over other simulators. Additionally, the use of patient specific data will allow the inclusion of all the contained information, further increasing the potential gain. Combining this with an accurate haptic interaction with the data will give a high immersion and high presence simulator.

This work will focus on the visualization module of the aforementioned neurosurgery simulator. This simulator aims to become a platform for training students in real surgery procedures. Students will get experience in the tasks involved in these operations, considerably reducing the number of hours they need to spend attending interventions. Additionally, it will realistically visualize and emulate patient-specific tissues, allowing residents and surgeons to use the simulator for rehearsal.

1.5 Objectives

The main goal of the project this dissertation is part of is the development of a realistic and immersive neurosurgery simulator oriented towards brain tumour resection. This neurosurgery simulator will feature realistic haptic feedback, high quality visualization and accurate simulation of the tissues' deformation. This simulator will be able to be used as a training platform for students or as a rehearsal platform for surgeons.

As a part of the project, the collisions and haptic interaction has been researched and implemented, as presented by (Echegaray, 2012). Said dissertation presented a high degree fidelity haptic interaction, for which the needed rigid and deformable collision detection modules were developed and implemented.

The main objective of this dissertation is the research and development one of the remaining modules, the visualization module. One of the most important parts of the simulator is the visualization module, which will provide the immersive sensation needed as well as the level of realism desired. Additionally, the integration of the present work with the work done by (Echegaray, 2012) will result in a simulator prototype capable of at least simple interactions with the brain. However, the development of the complete neurosurgery simulator falls outside the scope of this dissertation.

As it has been previously said, the visual quality and the accuracy of the models used in the simulator greatly affects the immersion and the transfer of knowledge. In order to achieve the best quality possible, medical images of the skull and brain of real patients will be used not only for the simulation but also for rendering.

For the accuracy needed for rehearsal of a procedure, data from the patient must be used. For the skull, a CT of the head offers a high contrast of the bone, which enables its use for a quality rendering of it. On the other hand, the brain needs to discern the different tissues present in it. For this purpose, a MRI of the head will allow its use for the visualization of the brain. Finally, the physical simulation module requires a tetrahedral mesh of the tissue, so it will have to be created from the medical images of the patient.

Consequently, the visualization module will have to be capable of rendering patient specific data with high quality and realism in real time. Additionally, it will be able to render the result of the physical simulation and interactions of the user. This visualization module has three main goals:

- The realistic visualization of patient specific rigid volumetric data such as the skull. This visualization must achieve real time rendering times whilst maintaining the highest possible quality. Additionally, it must be able to cope with rigid interactions such as drilling.
- The ability of volumetrically rendering deformable soft tissue, such as the brain, with a high degree of fidelity. Moreover, it must be able to perform said rendering and simulate the deformable tissue in real time.
- Both rendering types, the rigid and the deformable, must be seamlessly integrated in one unique simulator. Additionally it will have to communicate and interact with the rest of the modules such as the collision detection module or the haptic interaction module.

An additional fourth stage would involve the expansion of the simulator to a final neurosurgery simulator, but as it is been previously stated, it falls outside the scope of this thesis. 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 simulator would enable surgeons to practice actual operations.

A first stage of the development will focus on the two key elements needed to obtain the required realism: visual representation and interaction. Both elements will be optimized to offer the most realistic sensations possible whilst maintaining real time computing times.

The second stage will involve the addition of a physical simulation, along with the visualization of the deforming tissue. For this purpose a framework for deformable volume rendering will be developed, in order to maintain the highest quality possible.

This simulator's ability to use only volume data as input without data loss is a key advantage over state of the art simulators. In order to allow the use of the complete volumes, a framework for interacting with and rendering deformable volumes will be developed. Thanks to this framework the simulator will be able to run with real patient data, allowing its use for both medical training and surgery simulation.

1.6 Dissertation organization

This dissertation is organized as follows. Chapter 2 presents the present state of medical simulators. Furthermore, it gives some base information on image acquisition. Chapter 3 presents the most used volumetric rigid visualization methods, and presents various improvements and optimizations. Chapter 4 describes the problems presented by deformable volumetric objects and proposes a method to solve them. Additionally, a performance study of this method is presented. Chapter 5 presents the simulator prototype developed in this dissertation, as well as explaining which visualization methods have been used in each stage and why. Finally, Chapter 6 presents the conclusions of this dissertation and comments the possible future research lines.

Chapter 2

State of the art

No man is an island entire of itself,
every man is a piece of the continent,
a part of the main.
John Donne

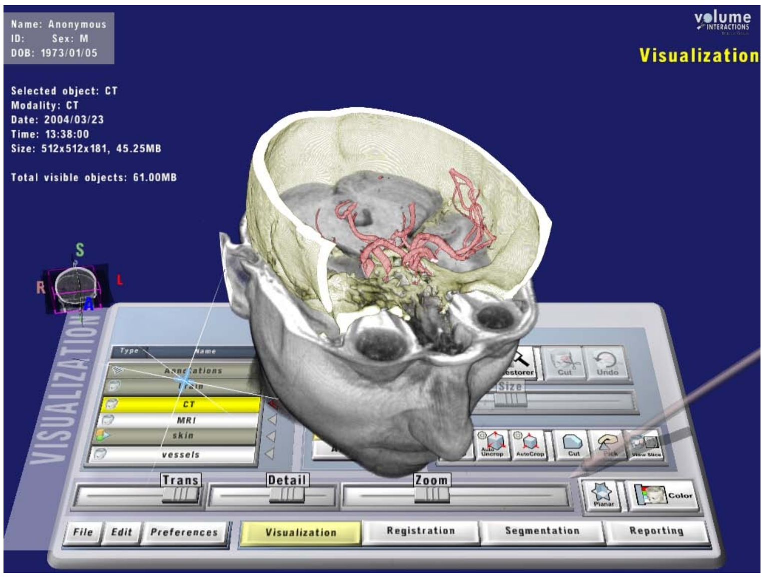
The use of virtual reality and haptic devices in medical simulation can greatly improve the skills of the user (either a student or a surgeon). Because of this, the use of simulation in medicine has been continuously growing since the 1990s. This thesis is focused in the visualization module of neurosurgery simulators. In order to better grasp the needs and gaps in this field the evolution of neurosurgery simulators is presented with a special focus on the visualization methods. Additionally, as many specialized fields are intertwined in this area, some helpful definitions and descriptions will be given.

2.1 Neurosurgery simulators

One early work was presented in 1994 (Gleason et al., 1994). In this work Gleason et al. used an augmented reality approach by superimposing a 3D virtual image of a polygonal surface extracted from a CT or MRI from the patient. To do so, the surgeon inspects the data from the patient and chooses the optimal approach for the surgery. Once the desired approach is set, a workstation is prepared in the operating room and the virtual data is registered with the real patients position using markers. After this the surgeon is presented with a combined image from the real data and the virtual data to use it as a guidance.

In a similar approach, Giorgi et al. proposed a computer assisted toolholder to help the surgeon control and check the orientation of different approach trajectories (Giorgi et al., 1994). By using a Virtual Reality system it guides a mechanical toolholder in a space of stereotactic neuroanatomical images by using high precision encoders on the arm are attached to the stereotactic frame.

Another early medical simulation framework is Dextroscope[1], presented in the mid-1990s, which focuses in neurosurgery applications. It uses rigid full volume rendering from CT and MRI images along with polygonal tools as it can be seen in Figure 2.1. Additionally, polygonal models can be used for certain anatomical models if needed. Dextroscope also has stereoscopic imaging capacity, and uses positional controllers to manipulate models with natural movements. However, this mechanism does not have haptic feedback. This framework can be used for planning, simulation or to inspect data from patients. However, the lack of haptic feedback and deformable volume rendering limits its utility.


Name: Anonymous
ID: Sex: M
DOB: 1973/01/05

Selected object: CT
Modality: CT
Date: 2004/03/23
Time: 13:38:00
Size: 512x512x181, 45.25MB

Total visible objects: 61.00MB

Figure 2.1: The visualization mode of the Dextroscope simulation framework.

[1]http://www.volumeinteractions.us/innovation.html

In the 2000's a company called *ImmersiveTouch* was created as part of a technology program at the National Institute of Standards and Technology[2]. As a result, they developed a medical simulation framework, also called *ImmersiveTouch* (Luciano et al., 2005).

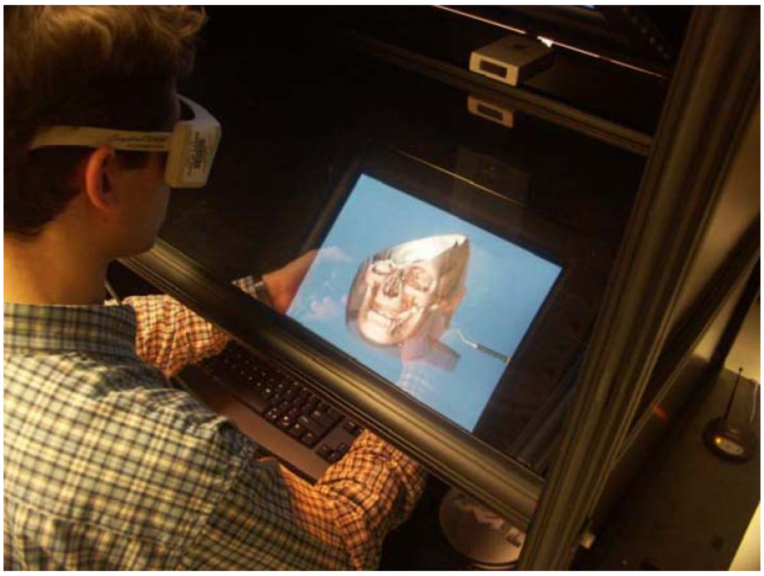


Figure 2.2: The prototype of *ImmersiveTouch*, showing a user interacting with the virtual workspace (Image from (Luciano et al., 2005)).

It uses an stereo visualization system, force feedback and head and hand tracking to create a virtual workspace where the user can interact, see and feel an object as if it were in their hands as shown in Figure 2.2. The objects are polygonal models extracted from volume data (usually CT and MRI) using the Visualization Toolkit (VTK)[3]. This conversion implies a loss of information as the rest of the volumes are not used and, as a result, lesser quality visualizations are achieved.

ImmersiveTouch has been used as the base for many simulators. For example, in the case of neurosurgery, Lemole et al. used it to create a training simulator of ventriculostomy and intended to expand it to be a trainer for different neurosurgical critical tasks (Lemole et al., 2007).

[2]`http://www.nist.gov/`

[3]The Visualization Toolkit `http://www.vtk.org/`

Later in the 2000's, thanks to the flexibility it offered, Dextroscope was used for many tasks. 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. However, the main improvement of VizDexter from Dextroscope is the ability to interact with the data while sharing the information with a larger audience. This allows a wide range of activities for the simulator, i.e. training by showing the work of an expert or to collaboratively create and discuss a surgical planning. Another work using Dextroscope presented a intracranial aneurysm clipping simulator (Wong et al., 2007). This simulator uses patient specific data and covers the whole clipping process: head positioning, craniotomy and aneurysm clipping.

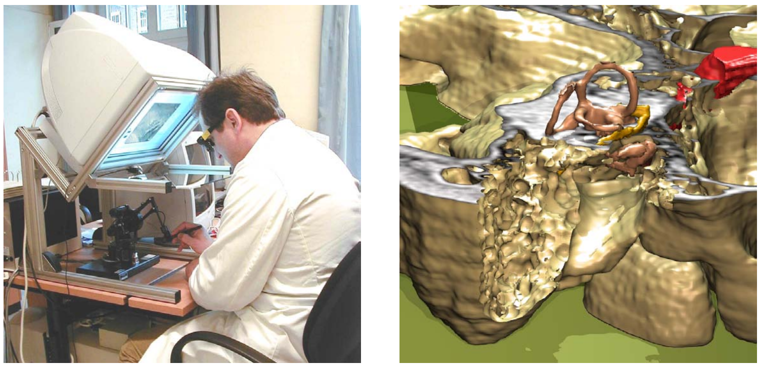

(a) (b)

Figure 2.3: Different parts of the simulator presented in (Pflesser et al., 2002; Petersik et al., 2002): (a) the physical system and (b) the visualization of the inner ear with multiple volumes.

Following the same basic scheme a simulator of petrous bone surgery was presented by (Pflesser et al., 2002; Petersik et al., 2002). This simulator also features a stereo visualization system along a 3 degree-of-freedom (DoF) haptic device as seen in Figure 2.3a. This work presents an advanced haptic rendering system as well as a volume visualization with adaptative sampling. To achieve the necessary rendering results 30 object models were extracted from a CT volume, discarding the rest of the volume. Afterwards it is rendered using a multi volume rendering as shown in Figure 2.3b.

Morris et al. developed a framework for temporal bone dissection

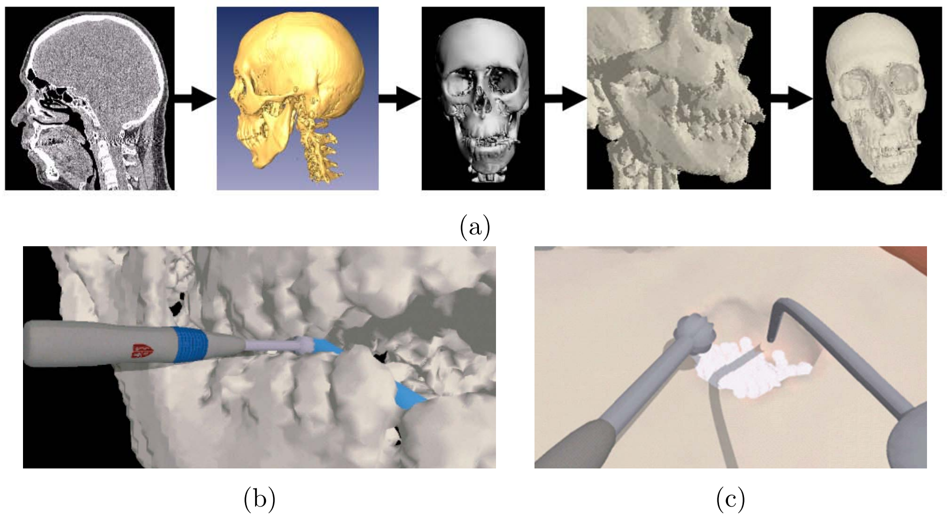


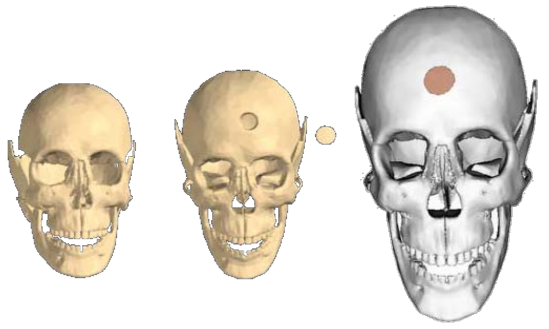
Figure 2.4: The resulting visualization of the developed presented in (Morris et al., 2005; Morris et al., 2006) in different stages: (a) surface extraction pipeline, (b) interacting with the skull and (c) drilling the bone.

(Morris et al., 2005; Morris et al., 2006). They used a hybrid data representation: bone is represented by volumetric data for haptic simulation of bone removal, and triangulated surfaces extracted from the volume are used for graphic rendering, resulting in the visualization shown in Figure 2.4. In order to achieve a good polygonal model, isosurfaces are extracted from the data obtained from CT or MR by using the Marching Cubes method (Lorensen and Cline, 1987).

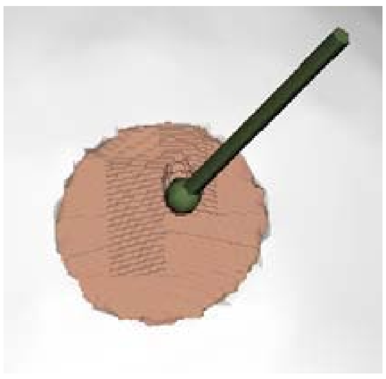
These isosurfaces are then capped using the 3ds Max[4] software package and a set of texture coordinates is generated on the isosurface mesh, a process that must be manually performed for each dataset. Afterwards, a flood-filling technique is used to build a voxel grid used for the haptic simulation. When bone voxels are removed as a consequence of the collision, the burred zone has to be re-tessellated because of the use of polygonal models. To do so triangles that contain the centres of the new surface voxels as vertices are added to the polygonal model.

With a similar idea, He and Chen presented a skull bone drilling

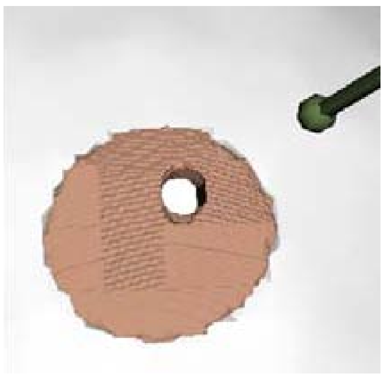
[4]http://www.autodesk.es/products/autodesk-3ds-max/



(a)



(b)



(c)

Figure 2.5: Simulator presented in (He and Chen, 2006). (a) the preprocessing, (b) the drilling process and (c) the polygonal models.

simulator (He and Chen, 2006). In this simulator a 6 DoF haptic device is used for the interaction and force feedback in contrast to the simulator presented by (Morris et al., 2005; Morris et al., 2006) that uses 3 DoF haptic device. In order to achieve a good drilling simulation, the haptic simulation is performed with a volumetric model of the skull.

However, only a part of the volumetric skull model is created to speed up the calculations. For this purpose, a volumetric model of the skull is created from the polygonal model and the space to be drilled is selected with the haptic device as shown in Figure 2.5a. After this the selected volumetric part and the polygonal model are integrated. The drilled volume is rendered

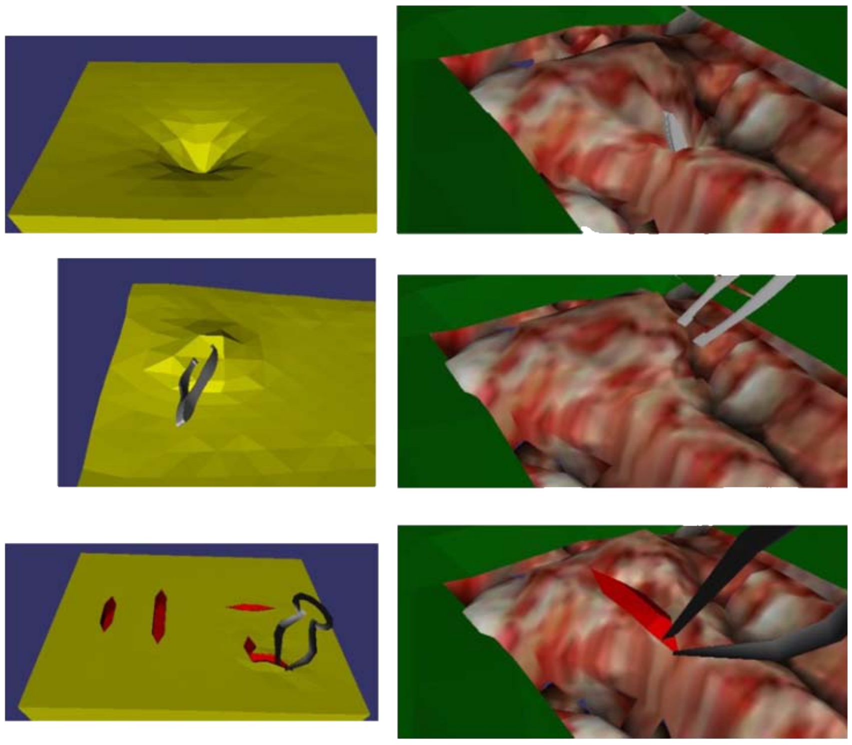


Figure 2.6: Different surgical actions like prodding, pinching and cutting in (Wang et al., 2006; Wang et al., 2007).

along a polygonal skull extracted using the Marching Cubes method as shown in Figure 2.5. For the haptic rendering the preselected area of the volume is used, forcing the user to interact only with that preselected area.

Another neurosurgical simulator was presented by (Wang et al., 2006; Wang et al., 2007), where a realistic modelling of the cutting and retraction procedures were used. This simulator used polygonal models to recreate deformable tissues. Realism was added by texturing the models as shown in Figure 2.6. However, the main focus of this work is to realistically simulate the cuts and retractions. For this purpose they developed a system which allows two handed procedures by using two independent 3 DoF haptic devices. By fusing the advanced interaction system with the physical model of the cuts, they achieved a high realism in the interaction with the simulator.

The visible ear simulator[5], created by (Sorensen et al., 2009), is a freely available simulator. It uses data from the Visible Ear digital image library, which was gathered through cryosections of a 85 year old woman's temporal bone. Thanks to this, a very high resolution model with the tissue's real colours can be used but limits its usage to the predefined volume. This largely reduces the use of the simulator as a training platform as differences between datasets, such as illnesses or deformations, cannot be easily introduced. This method leads to a very realistic visualization as shown in Figure 2.7. Additionally, this simulator allows the use of a 3 DoF haptic device for interaction and force feedback.

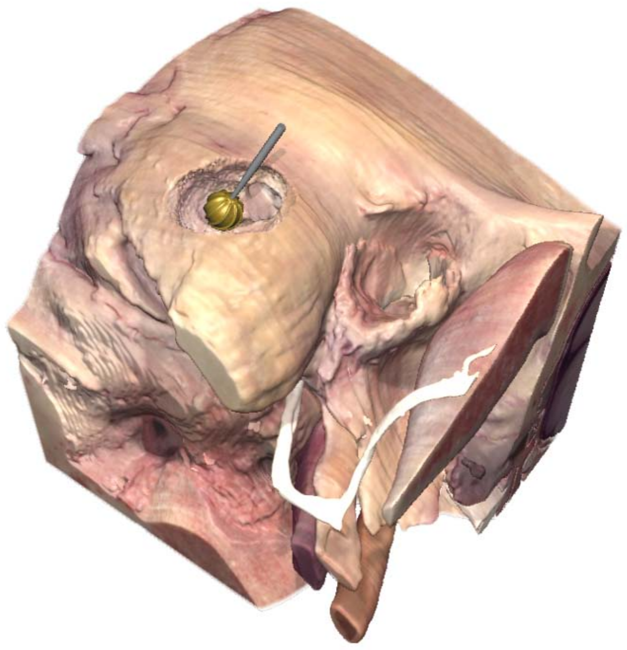
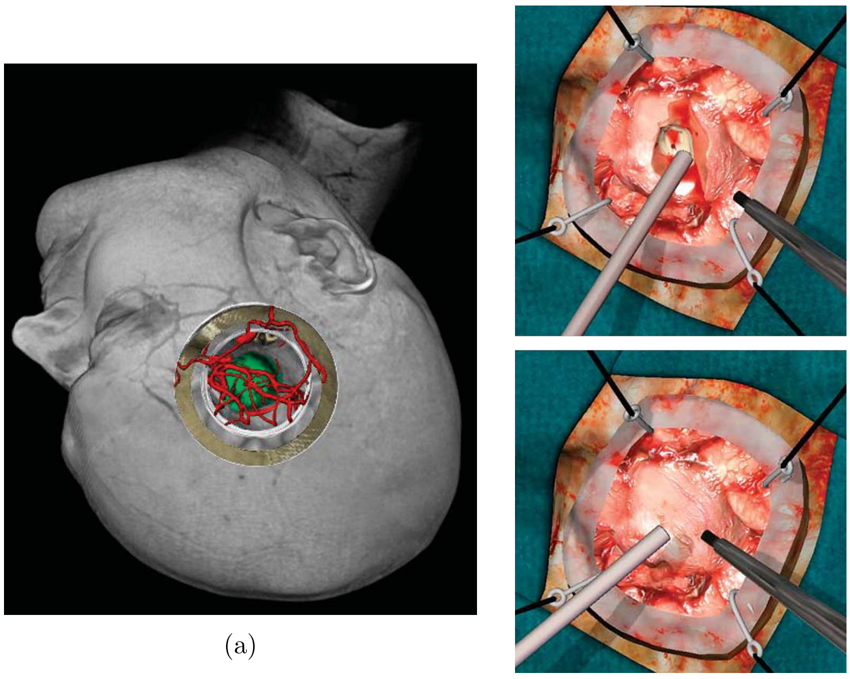


Figure 2.7: A snapshot of *The visible ear simulator*, showing the high quality rigid volumetric visualization.

More recently Delorme et al. presented *NeuroTouch*, a simulator for microsurgery training (Delorme et al., 2012). The setup of the simulator is an updated version of previously presented works, a stereoscopic visualization system combined with dual haptic devices (optionally 3 or

[5]`http://ves.cg.alexandra.dk/`

6 DoF). For the visualization in the planning stage the user is presented with a volumetric visualization of the head, shown in Figure 2.8a. But for the actual training task, in order to show deformable tissues, high quality polygonal models with high resolution textures are used (Figure 2.8b). Coupling the high quality visualization with a finite element simulation results in a realistic training simulation.


(a)

(b)

Figure 2.8: *NeuroTouch* offers: (a) a volumetric visualization in the planning stage and (b) a realistic polygonal visualization in the simulation stage (Images from (Clarke et al., 2013)).

This simulator has been recently expanded to enable its use for rehearsal and planning instead of just training (Clarke et al., 2013). For this purpose, a whole pipeline was developed where, starting form the MRI image, the desired tissues are segmented and registered and then polygonal models of the surfaces are extracted. For the planning stage, a volumetric visualization of the head is used. However, a polygonal visualization of the extracted brain surface is used for the interaction.

2.2 Medical Imaging

In order to achieve the highest possible accuracy in the models most simulators use medical image data, either directly or extracting surfaces from it. The origin and method of the image acquisition defines its utility for different simulators and uses. Because of this dependence in this section the main three data acquisition methods in medicine will be briefly explained: Computerized Tomography (CT), Magnetic Resonance Imaging (MRI) and Cryosection.

2.2.1 Computerized Tomography

The first non-invasive methods to allow the inspection of a living person was the X-ray. It allows physicians to inspect the state and the extent of the damage of a patient without much risk. But the X-ray lacks the perception of depth, making it difficult to use in certain areas. This hurdle was overcome when the X-ray computed tomography (CT) was presented in (Hounsfield, 1976).

The CT provides the density of the tissues codified in intensities, as shown in Figure 2.9. Additionally, these values have been studied (Lehmann et al., 1997) and after standardization are easily identifiable as shown in Table 2.1. As it can be observed it provides a high range for bones and some tissues, but the range for most soft tissues is short and overlapping. As a result, CTs are widely used when the objective is to inspect bones, but they are not well suited for soft tissue inspection.

Tissue	Value
Air	-1000
Lung tissue	-900 to -200
Water	0
Liver	35 to 60
Tumour	35 to 55
Bones	45 to 3000

Table 2.1: Some values of tissues in CT (Lehmann et al., 1997).

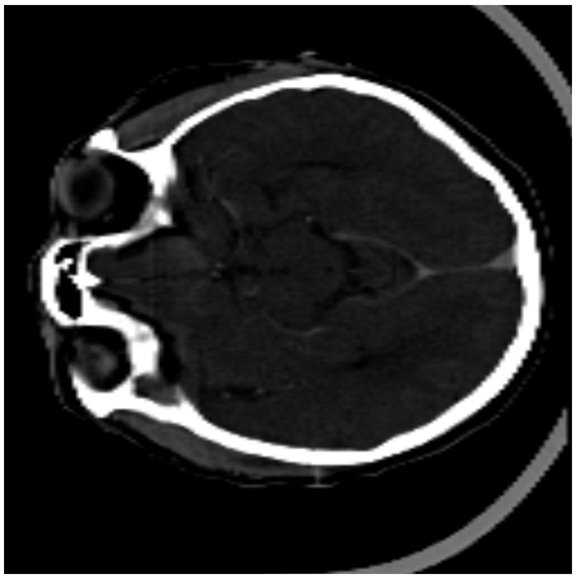


Figure 2.9: A slice of a CT of a human head. It can be seen that the skull is easily detectable in good contrast, as opposed to the brain.

2.2.2 Magnetic Resonance Imaging

Around the same time CT was presented, a different imaging technique was proposed (Lauterbur et al., 1973): Magnetic Resonance Imaging (MRI). Instead of X-rays, this technique relies on the magnetic properties of hydrogen atoms. The nucleus of a hydrogen atom is a single proton, allowing the interaction through a strong magnetic field. By studying this interaction of the hydrogen molecules in the body with the magnetic field, information about the tissues can be extracted (Mansfield, 1977).

The hydrogen atoms needed for this technique will mostly be located in water molecules in the tissues. As a result MRI is not very efficient for bones, but provides a larger contrast in soft tissues than CT, as shown in Figure 2.10. On the other hand in MRI data the tissue values are largely variable. Although a range of values for the tissues can be set large variations within that range occur even in the same dataset. For a radiologist this is not a large problem, as they are able to discern the tissues, but computationally this variability creates an added difficulty.

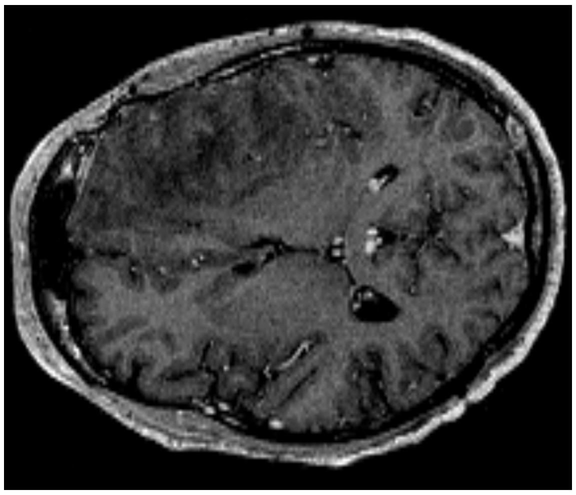


Figure 2.10: A MRI image of a human head, where the brain and the different tissues it is composed of are easily differentiable.

2.2.3 Cryosection

In medicine, cryosection is usually used to prepare tissue for its inspection in a microscope e.g. when removing a tumour it can be used to inspect it and check its properties. The procedure consists in embedding the tissue in a medium and then freezing it. This frozen block is then thinly sliced using a microtome obtaining a very thin slice of the tissue, usually just a few micrometers thick.

In order to create a volume this procedure can be slightly modified. In the modified procedure instead of just slicing the frozen block it is photographed, effectively creating a stack of photographs and thus a volume of the embedded tissue. This method created high quality images, as shown in Figure 2.11 but, it is clear that is not usable with a living patient. However, the high quality and the inclusion of colour information makes them desirable for certain simulators i.e. only training simulators which need an high quality anatomically correct model.

Accounting for the scarcity of the available volumes created with this method, several projects have appeared. One of the first projects was the Visible Human Project[6] from the University of Michigan where the whole

[6] `http://vhp.med.umich.edu/`

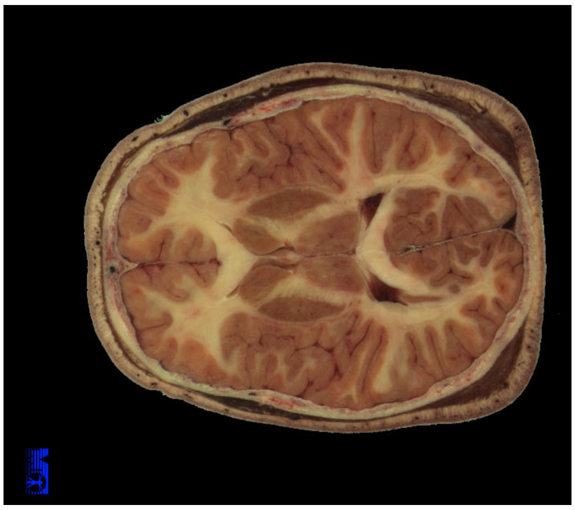


Figure 2.11: A cryosection of a head from the Visible Human Project.

body of a man was scanned with CT and MRI and then cryosectioned.

2.3 Discussion

This chapter has presented the evolution of neurosurgery simulators through the years, and the various solutions given to the visualization problems used in them.

As it has been shown, most simulators use volumetric visualization for rigid objects avoiding the loss of data caused by surface extraction. But due to the additional difficulty of interaction with the volume they switch to polygonal surfaces for interaction.

In more recent simulators, as the *The visible ear simulator*, volumetric visualization has been used for rigid interaction, i.e. drilling. This change has been fuelled by the increase in the computational power of graphics cards, which has rendered volumetric visualization capable of achieving interactive times with high quality rendering.

Nevertheless, even in the latest simulators such as *NeuroTouch*, the deformable interaction is still visualized with polygonal surfaces.

The extraction of polygonal surfaces for interaction creates a data loss, as it takes only the information from the selected surface. This means that the interaction will be performed with only that information, and that if more detail of the anatomy is wanted it has to be artificially created or extracted from a volume, having to select and extract all the anatomical features wanted in the simulation.

In contrast, if the whole volume was used, there would be no need to manually select all the features that are wanted to visualize, as all the information would be contained.

The purpose of this dissertation is to research the methods needed for the development of simulator capable of taking advantage of all the information contained within the volume. This means that it will need to interact with rigid and deformable volumes, calculating collisions and restoring forces realistically. In order to have a useful deformation it will need to have a physical simulation for the deformable volume. Additionally, it will have to volumetrically render both the rigid volume and the deformable volume to correctly visualize all this, .

Part II

Proposal

Chapter 3

Volumetric Visualization of Rigid Models

Part of this chapter has been published in:

Herrera, I., Buchart, C., and Borro, D. "Adding refined isosurface rendering and shadow mapping to vtkgpuvolumeraycastmapper". VTK Journal (Midas Journal), October, 2012.

Herrera, I., Buchart, C., and Borro, D. "Preserving coherent illumination in style transfer functions for volume rendering". In Information Visualisation (IV), 2010 14th International Conference DOI - 10.1109/IV.2010.16, pp. 43–47. 2010.

3.1 Introduction

Volumetric visualization of rigid models is widely used in scientific, industrial and medical visualization due to its high versatility and potential image quality. In the medical field, for example, it can be used for examination of patient data, planification of surgeries or simulation. In the context of this dissertation, rigid volumetric rendering will be necessary for the first stage of the simulator, as it can be seen in Section 5.2.

Because of its widespread use there are many algorithms and variations to fulfill the various requirements, from object order methods such as texture mapping or splatting to image order methods such as raycasting. Additionally, the high interest on volumetric visualization has triggered the development of various frameworks and libraries capable of handling these type of data. Nonetheless, volumetric visualization covers a high variety of data sources making the creation of a generic library a highly difficult task.

Although widely used, volumetric visualization also has some disadvantages that limit its usage in simulators. One of this disadvantages is its relatively high computational cost, which is one of the main reasons some simulators fall back to extracted polygonal models when using rigid volumes. However, thanks to the increase on computational power of commodity computers and algorithm- improvements, volumetric rendering can be used nowadays for high quality rendering with interactive frame rates in most modern PCs.

Even though the use of polygonal models simplifies and accelerates the rendering pipeline, it reduces the volume data to a mere surface thereby losing most of the information contained in the volume. Usually, the desired surface or information is set and the surface extracted and used. However, if the selection or the volume changes, the volume has to be reprocessed to extract the new surface. This forces such applications to store the volume and to reprocess it as soon as the selection has changed.

Additionally, as a result of the loss of information, transparency and translucency become nearly unusable with the polygonal models. This becomes an important issue in many stages of a simulator, especially in the planning stage, as these two visual effects can greatly improve the information given by the visualization to the user. For example, by using translucency, a surgeon can see both the organ and a tumour inside it, allowing a better understanding of the space. Another usage can be in training, as it can be used to help a beginning trainee see and learn the spatial relations.

Volume rendering, on the other hand, does not suffer from these inconveniences. As the whole volume is maintained, all the needed information is available. This avoids any processing needed for the visualization, since the information is used directly. This allows to seamlessly change the selection or visualization mode with nearly zero overhead.

Moreover, just by changing the visualization mode it can realistically render the tissue or highlight special zones or organs. All these advantages give volumetric rendering a high versatility and allows the user to perfectly adapt the image to the specific needs.

This chapter will first introduce various algorithms and works presented in the field of rigid volumetric visualization. Secondly, it will study the different needs and problems a volume rendering modules for surgery simulation faces and it will address these needs and issues by presenting different methods.

Additionally, the inclusion of the proposed methods in a well known visualization framework will be presented. This will allow non experimented users to use the methods presented in this chapter for the development of applications with volume visualization, such as surgery simulators or medical image visualizers.

3.2 Background

In most commons visualization pipelines, models are defined only by their surfaces modelled with polygons, usually triangles. This allows the highly specialised Graphic Processing Units (GPU) to efficiently render a high number of models in real time.

However, the basic primitive of a volumetric dataset is not a polygon, but a three-dimensional polyhedron. In the same sense that surfaces can be represented by different types of polygons, volume datasets can be defined with different types of polyhedra. A coarse classification of the representations modes is to define them by their structure. Structured volumes are those that follow a know pattern, whereas unstructured volumes are those that do not posses a regular layout. Figure 3.1 shows types of volumetric representation used for different purposes and fields.

As it has been explained in Chapter 2.2, medical imaging captures slices of the scanned object creating an uniform rectilinear volume. The basic primitive of this type of volume is called voxel, which is a rectangular hexahedra. As this type of structured volume is the most common, in this dissertation it will be referred simply as structured volume unless noted otherwise.

In order to visualize this type of data, the participation of each voxel in the final image must be calculated. For this purpose, the interaction of the light with the volume has to be calculated, which is a computationally

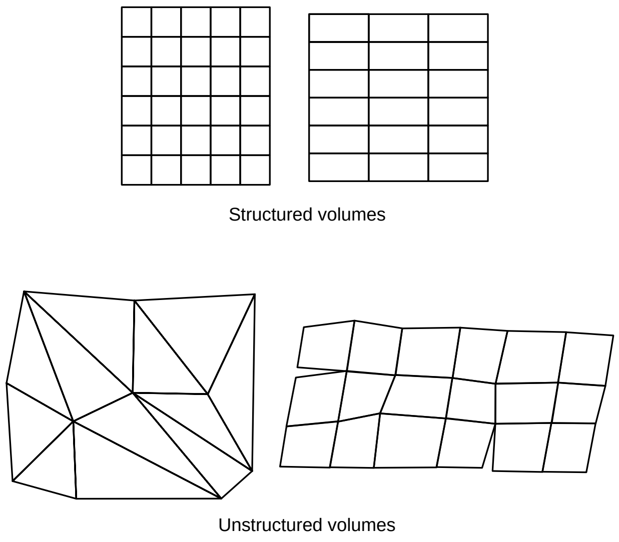


Figure 3.1: Two-dimensional representation of different volume types. The upper representations are a type of structured volumes, uniform rectilinear datasets, more often used in medical imaging. The unstructured volumes are more usually used in scientific visualization and simulations.

intensive calculation. Because of this, several simplified models have been developed:

- **Absorption only**: The only participation of the volume taken into account is light absorption. This would achieve a rendering of a only black volume.
- **Emission only**: The only participation of the volume taken into account is light emission. The resulting rendering would consist in a light emitting volume.
- **Emission-Absorption**: The volume absorbs and emits light. This achieves a good visualization with shadowing at a reasonable cost.
- **Single scattering**: The volume absorbs, emits and scatters light coming from outside light sources. Shadows are calculated with the exterior lights only.

- **Multiple scattering**: All interaction models are considered. This achieves the highest quality but it is the most time consuming.

The Emission-Absorption model is the most common one, as it allows a good visualization of the volume with a reasonable cost. All the methods presented in this work will use this models unless otherwise stated. This model is defined by the Volume-Rendering Integral (Equation 3.1). A complete and detailed derivation of this formula can be found in (Engel et al., 2006).

$$I(s_f) = I_0 e^{-\int_{s_0}^{s_f} k(t)\mathrm{d}t} + \int_{s_0}^{s_f} q(s) e^{-\int_{s}^{s_f} k(t)\mathrm{d}t} \mathrm{d}s \tag{3.1}$$

$I(s_f)$ is the final light coming out of the volume from the exit point of the ray, s_f. I_0 is the light incoming into the volume from the starting point of the ray, s_0. $k(t)$ is the light absorption function whereas $q(s)$ is the light emission function. To simplify, the repeating part of the equation can be defined as Equation 3.2. It can be seen that this equation calculates the amount of light that passes between the points s_1 and s_2 of the ray.

$$B(s_1, s_2) = e^{-\int_{s_1}^{s_2} k(t)\mathrm{d}t} \tag{3.2}$$

Basically, in Equation 3.1, the amount of the background light hitting the camera ($I_0 B(s_0, s_f)$) is calculated and then the accumulation of the light emitted by the volume along the ray (integrate $q(s)B(s, s_f)\mathrm{d}s$) added.

In order to use it in a real time rendering engine it must be further simplified, resulting in Equation 3.3. It must be noted that all volume rendering methods evaluate this integral in one way or another.

$$I(s_f) = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - \alpha_j) \tag{3.3}$$

The term α_i derives from Equation 3.2, and expresses the amount of light absorbed at that sample. From this, Equation 3.4 can be extracted which can be iteratively calculated from the camera to the final point of the ray (front to back order). The term C_i is the the colour of that sample multiplied by the opacity term of the same sample point, $C_i = Colour_i \times \alpha_i$ and the term C'_i is the colour accumulated up to that point. This is called opacity weighted colour, and it has been proven that its use is necessary when interpolating to achieve correct results (Wittenbrink et al., 1998). Similarly, the term α'_i denotes the α accumulated up to that point.

$$\begin{aligned} C'_i &= C'_{i-1} + (1 - \alpha'_{i-1})C_i \\ \alpha'_i &= \alpha'_{i-1} + (1 - \alpha'_{i-1})\alpha_i \end{aligned} \tag{3.4}$$

Additionally, by modifying this equation, it can be calculated from the initial point of the ray to the camera (back to front order) easily:

$$C'_i = C_i + (1 - \alpha_i)C'_i \tag{3.5}$$

One of the fastest methods for the CPU to evaluate the rendering integral is the shear-warp algorithm (Lacroute and Levoy, 1994). This method consists in projecting the volume into the viewing plane, for which the volume has to be sheared and warped to fit the viewpoint.

In a similar manner, volume slicing renders the volume slice by slice, using polygons to represent the slices and using the GPU for their renderization, significantly speeding up the process. The 2D texture mapping version uses volume aligned polygons textured with the volume, such as the ones in Figure 3.2. These textures are then bilinearly interpolated (Cabral et al., 1994). This method achieves very fast rendering times, but it creates artifacts. These artifacts arise from the alignment of the slices with the volume and the need to change said alignment depending on the viewpoint.

With the advance of the GPUs, 3D texture mapping made available the storing of the entire volume as a unique texture, and using it in arbitrary planes defined by 3D texture coordinates. Thanks to this method some of the artifacts created by the 2D texture mapping could be fixed, as the polygons does not have to be aligned with the volume. Instead, texture slicing with 3D texture mapping uses viewpoint aligned slices, efficiently reducing the artifacts and improving visual quality while maintaining a low computational cost. Additionally, as the polygons are aligned and thus the number of samples per pixel is exactly the number of contributing

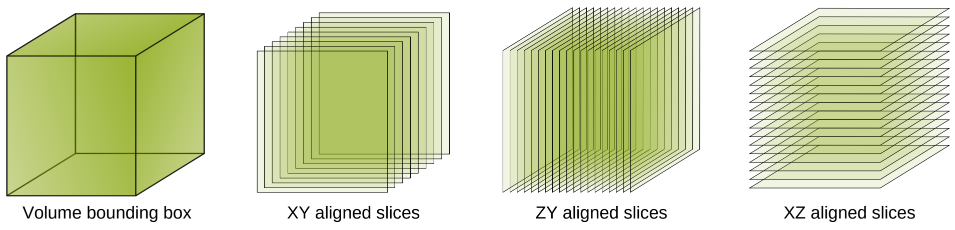


Figure 3.2: The three different alignments possible when using 2D texture mapping. The chosen alignment has to be changed in order to match as close as possible the viewpoint, which creates a flickering artifact.

polygons, it allows to control the number of samples per pixel (points of the volume evaluated) by increasing or decreasing the number of polygons used, as Figure 3.3 shows.

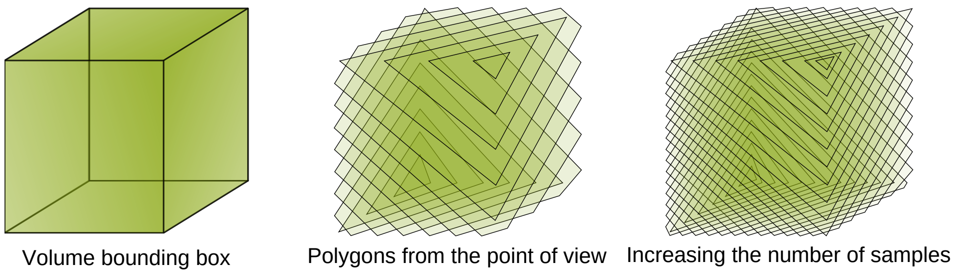

Volume bounding box
Polygons from the point of view
Increasing the number of samples

Figure 3.3: The view aligned polygons created to use with 3D texture mapping. These two versions map the volume at different sampling rates by duplicating the number of polygons from the central one.

Other object order volume renderings exist, such as splatting (Westover, 1991) or cell projection (Weiler et al., 2003). These method create different primitives (cell projection usually uses tetrahedra while splatting uses spheres) and project them onto the viewing plane. Although their rendering quality is among the lowest, their speed and the fact that they can be used with unstructured volumes make them interesting choices in certain cases.

The most straightforward algorithm to evaluate the volume rendering integral is raycasting (Levoy, 1988). This is a image order method that consists on shooting a ray from each pixel in the final image and evaluate the rendering integral by sampling points along the ray. This calculation is very costly, but with the the increase of the computational power of the GPUs along with the intrinsic parallel nature of the algorithm made it suitable for real time.

The first algorithms to render a structured volume through raycasting had to deal with the limitations on the conditions and loops of the GPUs at that time (Kruger and Westermann, 2003; Roettger et al., 2003). Said limitations increased the complexity of the algorithms and limited their speed. Nevertheless, these methods achieved high frame rates and good quality images. Modern GPUs have eliminated these restrictions while improving their computational power, with the resulting increase in the rendering speed.

Even though many methods exist, most ray casting algorithms have a common base pipeline. Figure 3.4 shows a flow chart of the usual ray traversal algorithm. It can be divided into two stages: ray creation and evaluation. In the creation stage the initial position and direction of the rays have to be calculated per pixel, taking into account if parallel or perspective projection is used.

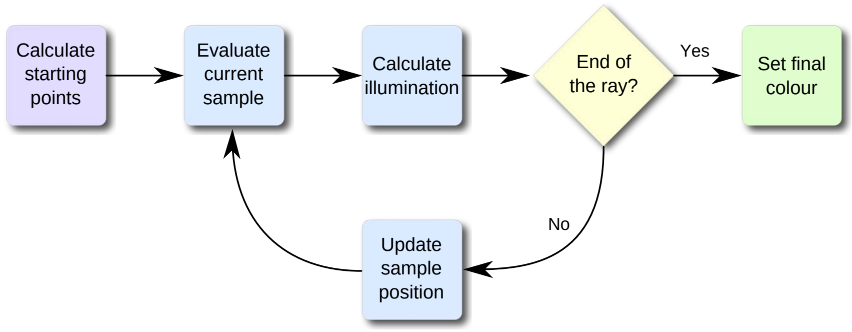


Figure 3.4: Flow chart illustrating the basic ray traversal pipeline. Note that these pipeline includes early ray termination and illumination.

A simple but effective way to create the rays is to render a proxy geometry (usually the bounding box of the volume) colour encoded with the corresponding volume coordinates (Kruger and Westermann, 2003). This creates the entry and exit points for each pixel, as shown in Figure 3.5.

In the ray evaluation stage the rays are traversed in order to evaluate the volume rendering integral. This evaluation is performed by sampling the

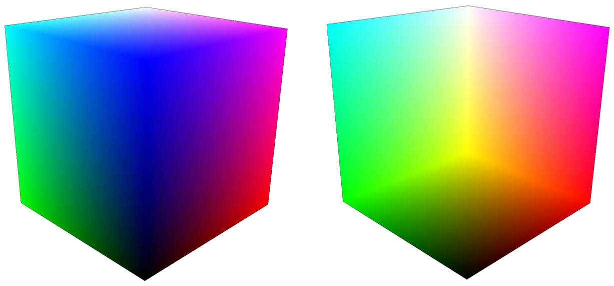


Figure 3.5: The colour encoded front faces (left) and back faces (right) of the proxy geometry of a volume in a perspective projection.

volume contribution one point (and optionally calculating the shading in the sample), accumulating it, and then going to the next sampling point. A very important part, in terms of quality and speed, is the distance between samples (sampling distance) also defined by the number of samples per distance unit (sampling rate). According to the Nyquist-Shannon sampling theorem, the sampling rate must be more than two times the sampling frequency of the sampled data. With a higher the sampling rate better visual results are obtained, but at the cost of a higher computational cost, thus an equilibrium must be achieved between quality and costs.

The ray traversal is usually done front to back order, as it allows to stop the ray when its maximum opacity has already been achieved thus avoiding unnecessary computations. Additionally, the use of raycasting also allows a variety of optimizations such as adaptative sampling (modifying the sampling distance depending on the materials properties) or empty space skipping (entirely jumping the samples in empty parts of the volume).

All these methods can achieve coloured images from the volume but, as it has been already said in Section 2.2, most volume acquisition methods only provide scalar volumes (intensity). This intensity has to be interpreted and the desired opacity and colour for each material decided, which is done using transfer functions and for the illumination lighting models such as Goraud, Phong, etc., are used taking the gradient at the sample point as the normal.

3.2.1 Transfer functions

Transfer functions assign optical properties to the values of the volume. They can be roughly categorized by the amount of data taken into account to evaluate one sample of the volume. For example, one of the simplest and most usual transfer function is one-dimensional which assigns an opacity and a colour to a single value. This simplicity enables easy manipulation and creation but also includes some disadvantages. These include the inability to discern some types of tissues if their values are not different enough and the difficulty to represent the interfaces between materials. Some examples of visualization with different one-dimensional transfer functions are shown in Figure 3.6.

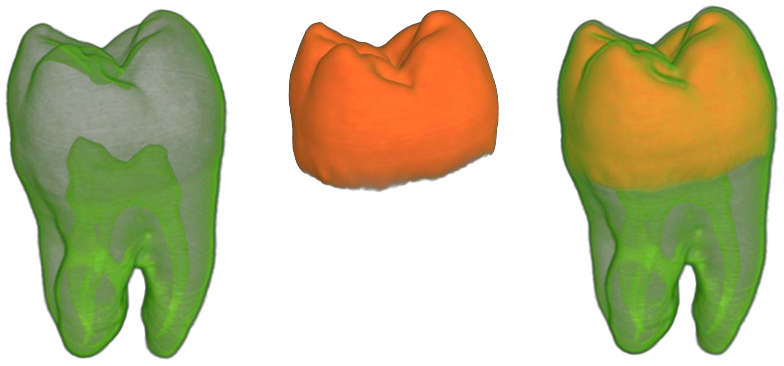


Figure 3.6: A tooth dataset rendered with different one-dimensional transfer functions. The leftmost image renders the interface between the tooth itself and the air, the image in the middle shows the hardest tissue in the tooth, and the last is the result of combining those two transfer functions.

By taking more data into account, these problems can be reduced or completely eliminated. A usual method to extract additional data from the volume is calculating the derivative. By calculating the first derivative (the gradient) more data is gathered, with the second more information about the interfaces between materials is added, and so on. Transfer functions that use this data are called multi-dimensional transfer functions. The most common is the two-dimensional transfer function where, in addition to the value, the magnitude of the gradient at the point is taken into account. This greatly enhances the differentiation between materials (Figure 3.7) and can potentially improve the image quality. On the other hand, by increasing the number of data used in the transfer function, the creation and modification become exponentially more difficult.

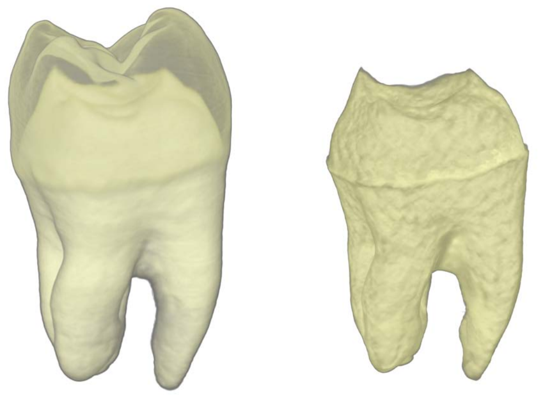


Figure 3.7: Trying to render the root of the tooth with one-dimensional transfer functions results in the image on the left. This is due to the fact that the interface between the tooth and the air has the same value as the root. The image of the right shows the root rendered with two-dimensional functions, which allow to differentiate between the air and the tooth by taking the gradient into account.

The creation of the necessary transfer functions is not a trivial task. Even the design of one-dimensional transfer function is not direct, and in many cases it derives into a trial and error method. As it has been said, the more data used in the transfer function the more detailed the rendering can be, but the more dimensions a transfer function has the more complex it is to create. Because of this, there is an ongoing effort in creating user interfaces that would allow an intuitive creation of multi-dimensional transfer functions (Correa and Ma, 2011; Wang et al., 2012). User aided design offers a great flexibility as the user can modify the created transfer function, generate it from scratch, or guide the algorithm in its creation. However, the need for the user intervention raises the knowledge required to use the application. Moreover, transfer function definition is not trivial and therefore it may become into a trial and error method. For example, to overcome this, Correa and Ma created a user guided interface that semiautomatically creates transfer functions and iteratively refines it through user interaction (Correa and Ma, 2011).

Another alternative is to remove the need for user interaction altogether, by generating the transfer functions completely automatically. This automatic transfer function generation usually focuses on the features (Maciejewski et al., 2009) or contour (Zhou and Takatsuka, 2009) of the volume. These kinds of methods face a major challenge when dealing with the flexibility of the method, as the transfer functions needed for different fields are very diverse, and even inside the same task the transfer function may differ completely for each volume. By detecting the desired visualization and the appropriate transfer function, these methods make volume visualization available for a wider population, as it does not require any additional skill to use it. More recently other methods have been presented to create the transfer function using information about the target visualization, for generic applications (Ruiz et al., 2011) or for specific applications (Lathen et al., 2012).

3.3 Improvements on Rigid Volumetric Visualization for Simulation

There are many rendering modes usable in volumetric visualization such as maximum intensity projection (MIP) where only the highest value on the ray is shown, or full volume rendering (FVR), where the ray is evaluated and its transparency and colour accumulated which is the most common. Another method is called direct volumetric isosurfaces, which can be used to visualize completely opaque materials without extracting the surfaces.

FVR raycasting offers the highest quality rendering among the volumetric rendering methods, but it is not free of rendering artifacts. However, thanks to the high flexibility offered, many improvements can be made in order to correct the artifacts or at least mitigate its effects. Additionally, many effects can be added to improve the quality and usability of the visualization.

The most common artifact in volume rendering algorithms is the woodgrain artifact, shown in Figure 3.8. This artifact is a direct result of the used sampling distance, and the bigger this distance is the more visible it becomes. It is not, however, possible to fix it by just increasing said distance, as it is also the result of the discrete nature of the algorithms.

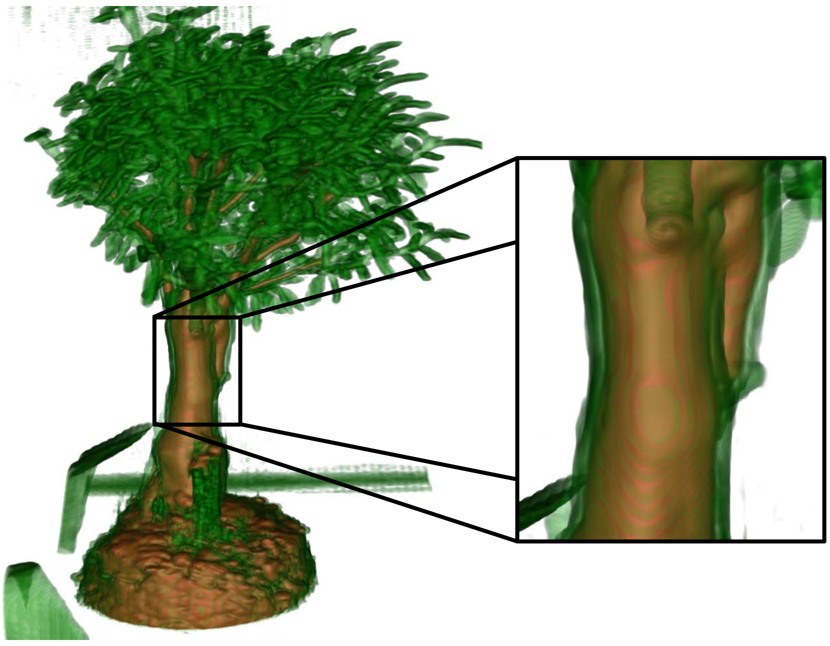


Figure 3.8: The bonsai CT dataset rendered with a translucent transfer function for the bark. In the close up image the woodgrain artifact is easily visible due to the sudden colour changes in the transfer function.

In the case of FVR raycasting, the problem arises from the interfaces of the materials and the discrete sampling of the rays. When a ray arrives to the frontier between two materials, it will evaluate a sample outside and another inside the volume (as expected). Additionally, the neighbouring rays will also do so, but if the rays and the object are not perfectly perpendicular in all the image the distance at which the ray will meet the interface will vary. This, coupled with the fact that all rays are sampled at the same regularly spaced points, creates the woodgrain artifact. Figure 3.9 shows an image illustrating this effect.

As it can be seen, it is not only the change from one material to the other that creates the wood grain like pattern but the regular sampling of the ray. It is evident that it is not possible to avoid the interface between the materials, so the problem to be tackled is the regular sampling of the rays.

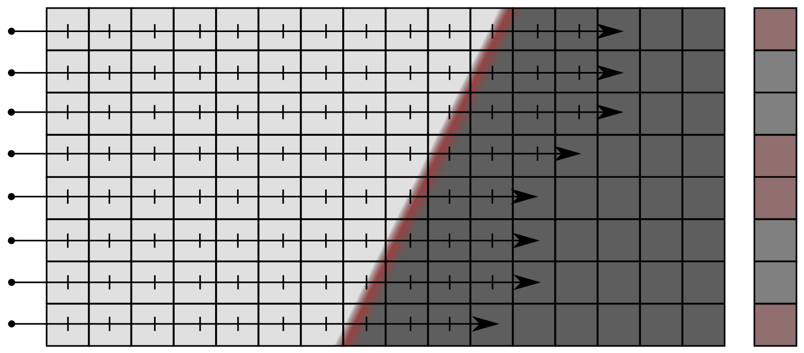


Figure 3.9: A simplified illustration of the raycasting algorithm, shows an example of how the woodgrain pattern is formed. As the rays start from the same distance and the sampling distance is regular, a pattern is created in the resulting image.

In order to disrupt the pattern a very useful tool is stochastic jittering (Engel et al., 2006). This method introduces a random variation in the starting point of the ray (in the direction of the ray) producing a random variation of the sampling points between the rays, as shown in Figure 3.10.

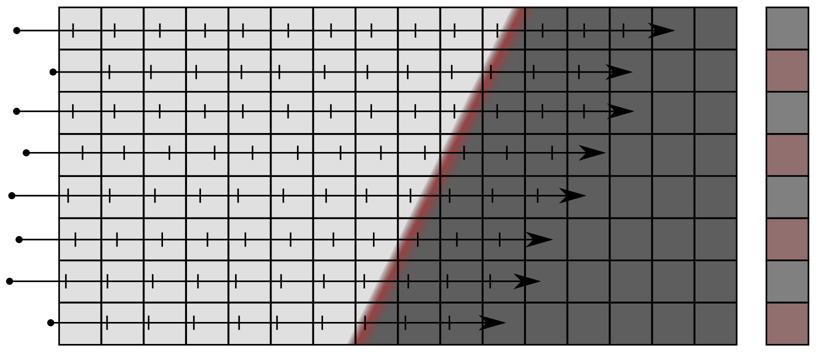


Figure 3.10: The inclusion of a random variation in the starting point converts the woodgrain pattern into a more random one. It should be noted that such variation is only included in the direction of the ray.

As a result of this controlled randomization the woodgrain pattern is disrupted and transformed into a random noise pattern. Due to the difficulty the human eye has to detect random noises in contrast to its ability to detect regular patterns, the woodgrain artifact disappears and

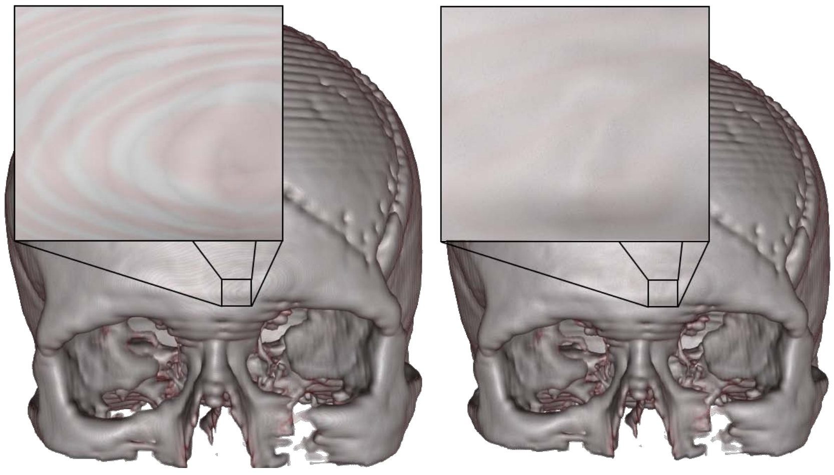


Figure 3.11: Comparison of a skull rendered without (left) and with (right) stochastic jittering.

the introduced noise becomes barely noticeable, as it can be seen in Figure 3.11. This method is, of course, limited with the used sampling distances as magnitude of the noise needed to efficiently mask the artifact grows along it. But the increase noise magnitude also makes it more noticeable eliminating its advantages. For this reason the magnitude of the noise is usually set beforehand and kept constant regardless of the sampling distance.

3.3.1 Direct volumetric isosurfaces

In the simulator prototype presented in this dissertation, there will be two volumes, a rigid skull and a deformable brain. The data for the skull comes from a CT and, as aforementioned, the value for the skull is very well defined.

As the skull is completely opaque and well defined direct volumetric isosurfaces can be used, providing a better quality and performance than FVR. This method renders the isosurface using a modified version of the ray traversal in FVR raycasting. The samples in which the value is smaller than the given isovalue are completely discarded, and the renderer proceeds to evaluate the next sample position. Samples with a value equal or bigger than the isovalue stop the ray traversal. The rest of the calculations, such as lighting or shadows, are done normally. In this rendering mode the opacity property is not given by the transfer function but is set to be completely opaque.

As a consequence of these changes, the number of calculations done per ray is drastically reduced, giving a huge performance boost. However, due to the discrete nature of the sampling and the volumetric data, some errors appear when the sample location does not exactly correspond with the isosurface. This artifact arises for the same reasons woodgrain artifact appears, but even though stochastic jittering is an option, a better solution exists for this case.

As the value of the surface to show is known, instead of inserting random noise to mask the error, the exact position in which this value is found at the ray can be searched. This method is called intersection refinement (Hadwiger et al., 2008), and consists in iteratively searching for the correct intersection point of the ray with the isosurface. This is performed by a binary search algorithm, shown in Equation A.1. This search is not usually performed until the desired precision is met, instead it is iterated a fixed number of times. Usually with as few as four iterations a very good result can be achieved.

$$X_{new} = (X_{next} - X_{prev})\,\frac{isovalue - V_{prev}}{V_{next} - V_{prev}} + X_{prev} \tag{3.6}$$

The iteration starts when a sample is evaluated to have a value higher or equal to the isovalue. Then Equation A.1 is evaluated, X_{prev} being the previous sample point, V_{prev} its value, X_{next} the point where the isovalue was surpassed first and V_{prev} its value. From this a new point is calculated, X_{new}, and its value, V_{new}, fetched. Then the usual binary search step is performed, as shown in Equation 3.7, and then it proceeds to the next iteration.

$$\begin{cases} V_{new} \geq isovalue & X_{next} = X_{new} \\ V_{new} < isovalue & X_{prev} = X_{new} \end{cases} \tag{3.7}$$

As a result, the quality of the isosurface rendering is greatly increased, as it can be seen in Figure 3.12, enabling its use in high quality visualization as a replacement for extracted polygonal isosurfaces.

Even though direct volumetric isosurfaces present evident

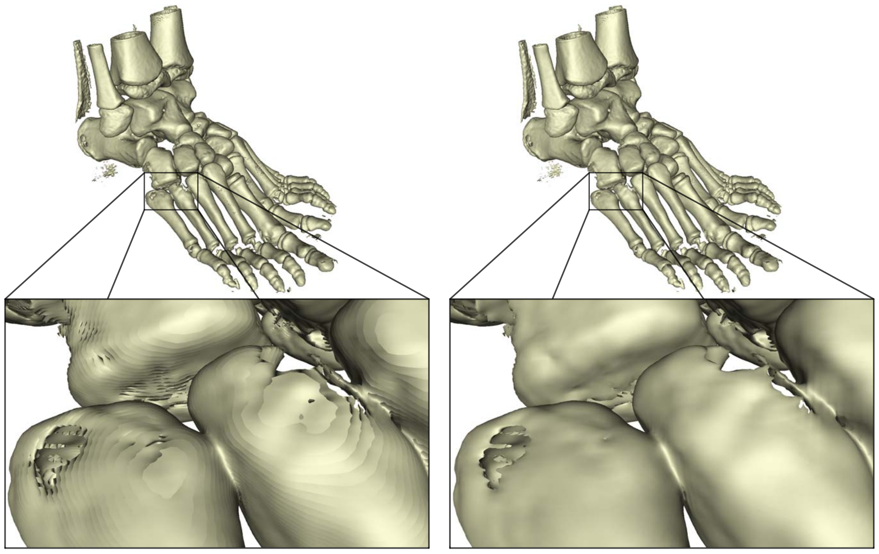


Figure 3.12: A CT of two feet rendered with direct isosurface rendering. The image on the left is rendered without intersection refinement whereas the image on the right is rendered with intersection refinement.

computational advantages over FVR, it is possible to perform it using specially created opacity transfer functions. However, refined intersection isosurfaces have an additional capability: is to achieve very good quality images even with high sampling distances. This can be used to further increase the performance of the rendering, and this optimization, along with the intersection refinement, gives direct volumetric isosurfaces a clear advantage over FVR. Figure 3.13 shows a piggy bank volume rendered with different sampling distances using both FVR and direct isosurface rendering. It can be easily seen that the isosurfaces maintain a very high quality even with very large steps, in contrast to FVR.

Furthermore, an additional optimization can be performed when the sample distance is small enough, that is, if the new position will still be contained between the previous and the next voxel. Taking into account the fact that the values between voxels are usually interpolated, the change of the values from the first sample to the next it is know to have been interpolated. Thus, if said conditions are met, with just one evaluation of

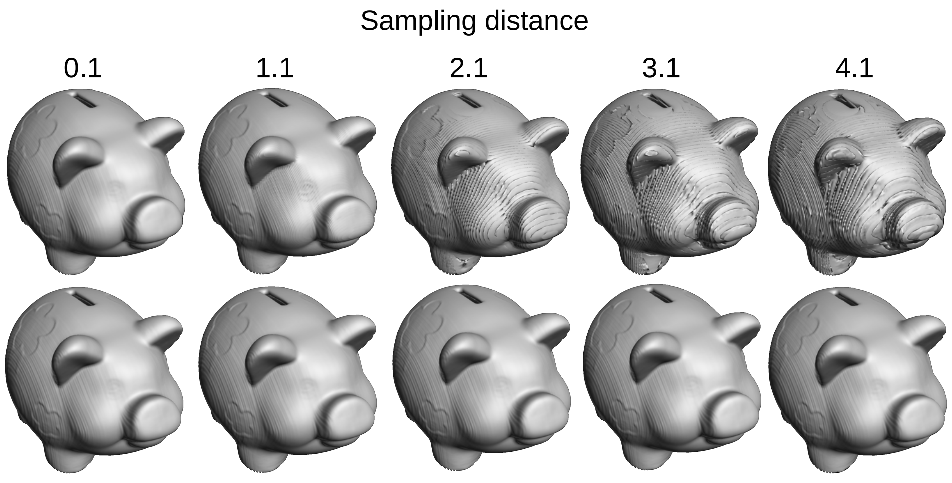


Figure 3.13: The piggy bank dataset illustrates how isosurfaces with intersection refinement (down) maintain quality with the increase of the sampling distance, while quality of the isosurface with transfer functions (up) steadily decreases.

Equation A.1 the correct point is met, as it will calculate the point in which the interpolation achieves the desired isovalue. This optimization is specially important in older systems, where performing a condition in the GPU is computationally costly, as it avoids the evaluation of Equation 3.7.

The resulting pipeline of ray traversal with isosurfaces is shown in Figure 3.14. As it can be seen the modifications needed to accommodate this pipeline are minimal and it adds several improvements, as it has been explained beforehand.

3.3.2 Differences with polygonal rendering

As it has been already explained, volumetric rendering has some key advantages over polygonal rendering of extracted surfaces. These advantages arise mainly from the fact that extracting the surfaces from a volume ignores the rest of the information, whereas volumetric rendering allows the use of all the information contained within the volume. This allows the use of transparencies and translucencies naturally and with high quality renderings, as well as easy on the fly changes on the optical properties of the materials.

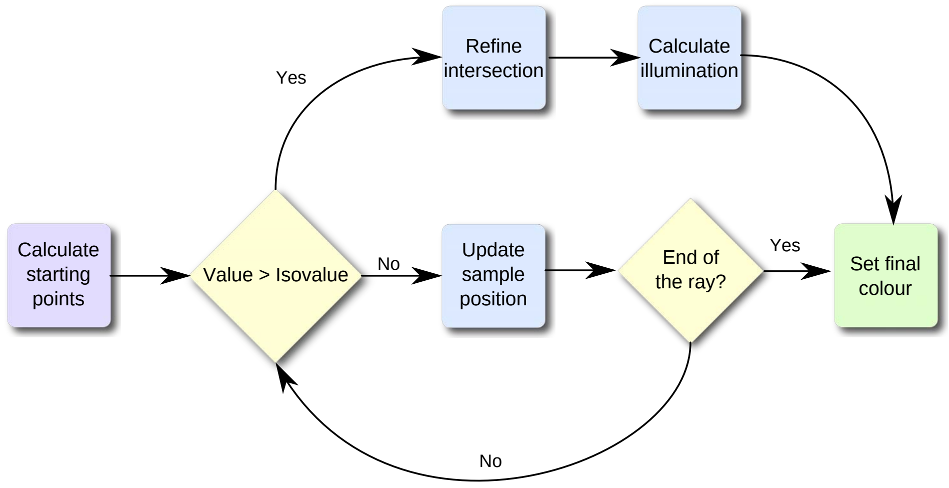


Figure 3.14: Flow chart illustrating the modifications made to the basic ray traversal pipeline for direct volumetric isosurfaces with intersection refinement.

Additionally, even when using volumetric isosurfaces where transparencies and translucencies are no longer an advantage, the use of raycasting allows pixel accurate isosurfaces regardless of the viewpoint or distance to the object. This is due to raycasting being an image order rendering algorithm in contrast to polygonal rendering, which is an object order algorithm. Thanks to this, the only limit on the image comes from the volume resolution. Polygonal rendering, on the other hand, is constrained by both the volume resolution and the extracted surface's polygon count and distribution. This difference can be observed in Figure 3.15, where a direct volumetric isosurface is compared to an extracted polygonal surface.

But these advantages do not come without problems. For example, while the hardware for rendering (GPU) is very specialized for polygonal rendering and it is very efficient in this task, volume rendering algorithms are forced to find ways to take advantage of said specializations in order to take full advantage of the hardware. In the last years, however, the fast increase on the GPUs capabilities and the generalization of their pipelines is steadily closing this gap.

One of these problems arise when mixing polygonal and volumetric objects, and comes from the fact that rendered volumes do not have a

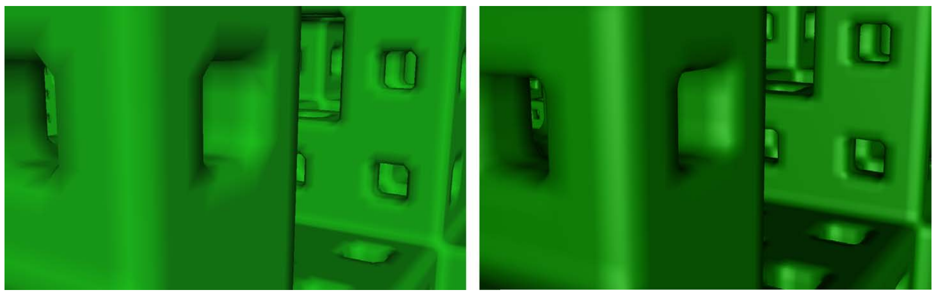


Figure 3.15: A close up view of a Menger Sponge dataset. The image to the left shows the render using an extracted isosurface with nearly eleven million triangles. The image of the right shows the direct volumetric isosurface rendering of the same dataset.

specific depth per pixel as polygons do. Because of this the usual depth culling methods do not work, and it must be performed inside the ray traversal. The most common solution is to first render the polygonal objects and, with the created depth map, test each sample in the ray against it to ensure that the samples behind any of the objects are properly culled. This method works correctly as long as there is only one volume at a time in a specific point, as the resulting depth from one volume rendering cannot be correctly used to render the next. Instead, they must be rendered at the same time, increasing the complexity of the method and reducing its scalability.

In the case of volumetric isosurfaces, however, there is a clearly defined depth value as in a polygonal rendering. Thanks to this, just by rendering them after the polygonal objects but before the full volume rendered objects any number of volumetric isosurfaces can be rendered correctly without any changes in the algorithms.

Another problem volumetric visualization has in comparison with the usual polygonal rendering is the calculation of the shadows. As volumetric objects are no longer mere surfaces and can be translucent, their shadows can be translucent and the calculation of both the shadows cast and received is more complex.

A very common shadowing algorithm for polygonal rendering is called shadow mapping. This method renders the objects that will cast shadows from the light's perspective and stores the depth map. Then, when the main rendering is performed, this map is consulted per each pixel and the transformed depth is compared to the depth of the current pixel, casting the shadows in the desired surfaces. However, as it has been said, this algorithm cannot be directly used to cast shadows from volumes, but modified versions are available.

Several approaches exist to this problem and can be grouped in three main types: no volume cast shadows, hard volume cast shadows and soft volume cast shadows (Hadwiger et al., 2008). The first approach just ignores the volumes when casting the shadows, but allows the volumes themselves to receive shadows. Although it is a rather simple solution it is very well suited to a vast range of applications, as many of them only have one volume in the scene. Additionally, that volume will usually be the centre of the visualization, and while the shadows cast by a polygonal object may be important, the shadows cast by the volume itself are not as relevant. This approach also allows to completely avoid an additional volumetric renderization. This is the case of many medical simulators, where it is common to have one main volume and polygonal instruments interacting with it.

The hard volume cast shadow methods avoid the problem by managing volumes as if they were not translucent, thus the usual shadow mapping algorithm can be used just by including a modified volume rendering. This volume rendering is simplified by ignoring transparency and treating the first hit as completely opaque. This creates hard shadows for the volume that do not correspond to the actual transfer functions, but enables the use of shadows without greatly increasing the cost.

The soft volume cast shadow methods completely changes the shadow mapping technique, taking into account the whole volume and its translucency. The most common method in this category is the deep shadow maps algorithm. This algorithm is similar to the normal shadow map algorithm: it renders the scene with the shadow casting objects from the lights point of view, but instead of creating a single map it creates a set of maps at different depths with the absorbed light up to that point. This creates a volume describing how much light arrives at each depth. Although this method creates the highest quality shadows, the need of a full volumetric render for the creation of the deep shadow map in addition to the main volume render as well as the increase of the cost of checking the map makes this method computationally very expensive.

In the case of the simulator prototype presented in this dissertation, the

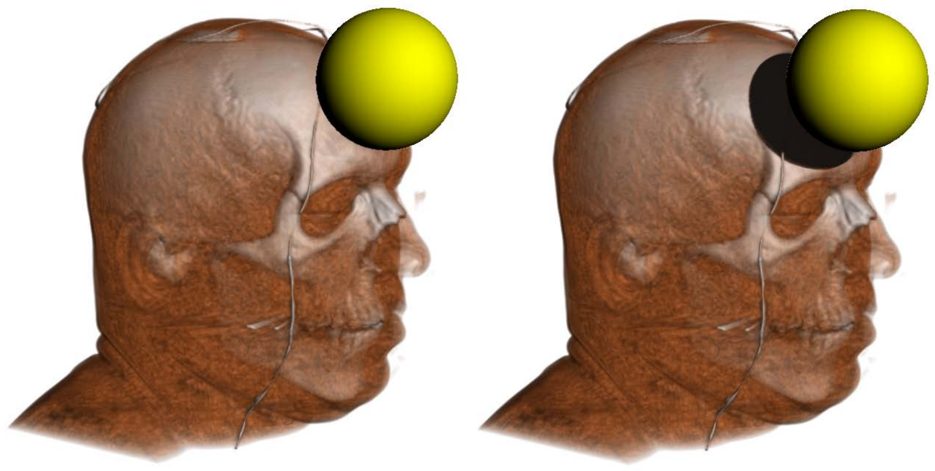


Figure 3.16: A comparison of the rendering of a sphere without (left) and with (right) casting shadows onto the volume. It is easily observed that the shadow greatly increases the perception of the spatial location of the sphere.

main focus of the visualization are two volumes, the skull and the brain. Taking this into account, and the fact that the brain will only be visible through the holes drilled on the skull, the first option is perfectly valid, as the shadow from the skull onto the brain will be mostly out of sight but the shadow of the tools on the skull and brain are necessary clues that increase the depth perception as it can be seen in Figure 3.16.

The pipeline resulting of applying all aforementioned improvements and optimizations shown in Figure 3.17 is able to efficiently render volumetric objects along other polygonal objects with high quality.

3.3.3 Style transfer function designer

The creation of the transfer functions is not a trivial task, specially when multiple types of visualizations, such as illustrative visualization or realistic visualization, are involved. The mixing of these two kinds of visualization increases the complexity of the creation of the transfer functions, as they can be as different as to require different illumination calculations.

In order to simplify the creation of illustrative transfer functions, style

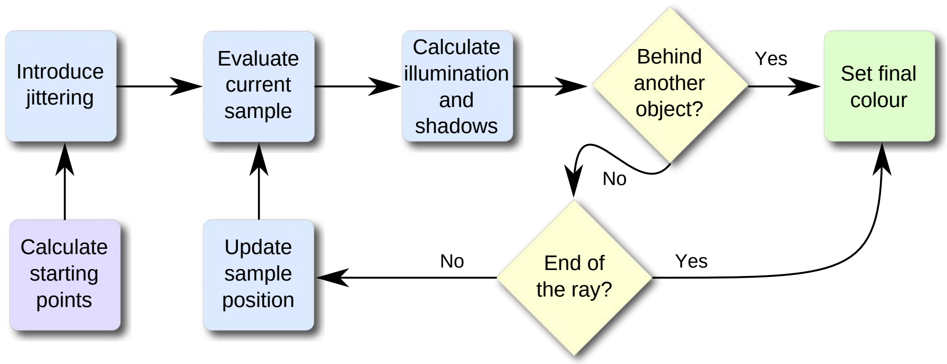


Figure 3.17: Flow chart illustrating the complete pipeline for FVR, including stochastic jittering, shadows and intersection with polygonal objects.

transfer functions were proposed (Bruckner and Groeller, 2007), where the transfer function maps a sphere map (instead of just a colour) to the voxel density and computes the final colour with the style and the voxel's normal (approximated with the gradient of the voxel). (Rautek et al., 2007) continued this work, adding a semantic layer to the style transfer function definition simplifying its design. Furthermore, (Herghelegiu and Manta, 2008) proposed the use of style transfer functions to achieve illustrative rendering through differently lit sphere maps.

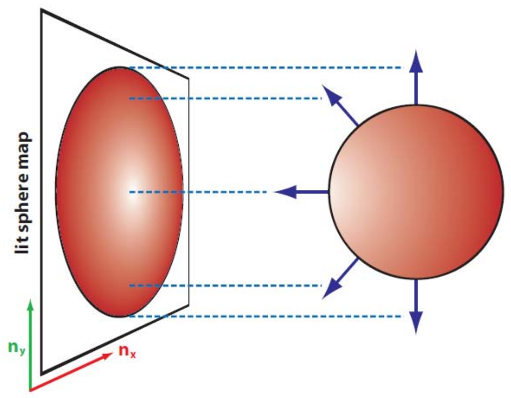


Figure 3.18: Lit sphere shading in style transfer functions (Bruckner and Groeller, 2007). It can be seen how the illumination model for the sphere is stored in the final style.

Thanks to styles, the illumination of different artworks could be transferred to the rendering, as styles implicitly store an illumination model. Once the style of the corresponding voxel has been decided, the gradient is used to create an approximation of the normal in eye space coordinates and then to choose the colour of the voxel from the map (Figure 3.18), effectively transmitting its illumination model to the final rendering.

In this way, the illumination from the sphere is transferred to the corresponding part of the volume, each one with its implicit illumination, and so the need of illumination calculations is removed from the raycasting. On the other hand, as every style has its own illumination model, and multiple styles are usually used in one volume, there is some risk of having styles with incoherent illumination models. This may be acceptable or even desirable in some styles of illustrative rendering, but in most cases, and especially in realistic rendering, it is utterly important to have a coherent illumination model, i.e., each one of the styles, even having their own independent models, must be coherent with all the other styles. In order to maintain the coherency manually, the user would need to use external programs and require additional skill in order to change the pictures used in the styles while maintaining everything with the same implicit model.

All this work is based on predefined styles, adding different layers to simplify their use and modification. In this dissertation a style editor is presented, which will allow the user to create the styles without having to rely on predefined styles (that are usually edited in third party applications such as photo editors). Using this approach, the user can easily manipulate the lighting model, which is maintained coherent through all the styles by the editor, or change the optical properties of the material by changing the base colour or applying a texture. The editor also allows applying different effects to the style as contour highlighting (Bruckner and Groeller, 2007) or central transparency (Buchart et al., 2009). Moreover, thanks to the implicit storage of the illumination information, the mixing of illustrative and realistic visualization is greatly simplified.

The proposed interactive designer gives the user total control over the styles and at the same time ensures that the styles will maintain a coherent illumination (independently of the changes made to them). Doing that, the time and skill previously required are dramatically reduced. Additionally, as styles are not simple pictures, but stored information (texturex, effects, colors, etc.) prepared to be rendered when needed, the illumination of the

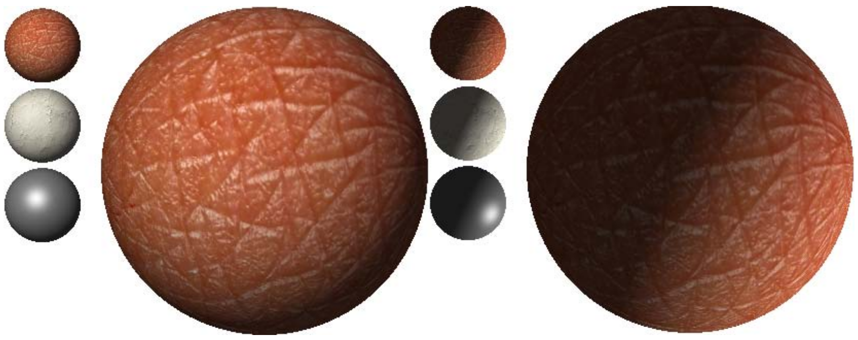


Figure 3.19: Sphere maps differently illuminated created with the proposed designer. It can be seen that the different styles (the small spheres) are coherently illuminated and maintained.

styles can be changed interactively, making it possible to interact with the light as easily and intuitively as with normal transfer functions and direct illumination (Figure 3.19).

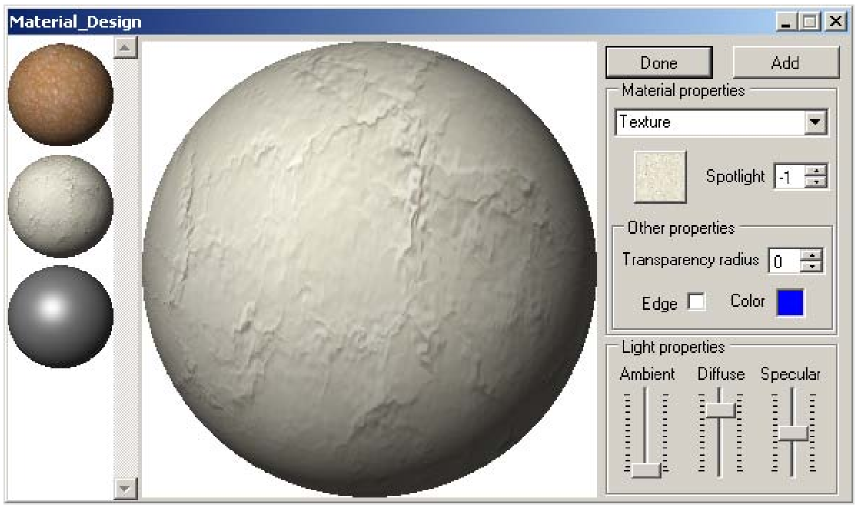


Figure 3.20: The proposed style designer with three defined styles for skin, bone and metal.

The designer enables the interactive use of style transfer functions through an easily usable style design and modification interface. The interface (Figure 3.20) shows the existing styles in the library, as well as the specific information of the selected style. Nevertheless, the general property that all the styles share, the direction of the light, can be modified independently of the selected style (using a click and drag operation on the selected sphere). While the modification of the light's direction is done in the selected style, the changes will be reflected equally in all styles (shown in the left panel). In this way, the user can have a preview of the result without having to select the other styles.

The user can choose, through the editor, a texture for the selected style, so a real photo of the material to be rendered can be used. For example, in medical visualization, the user could import a photo of a part of a bone in the style to be used to render the skull. Also, the user can select a plain colour, allowing an easily controllable illustrative style that, even though not realistic, it will be illuminated in the same direction as the other styles.

Albeit uncommon, the user might not want the illustrative style to be shaded in order to be more strongly highlighted. The editor makes this possible by means of the direct control of the way the light interacts with the styles. As it can be seen in Figure 3.20, the editor also gives the user power over the styles light's ambient, diffuse, and specular values. By modifying these values, the user can create shaded or not shaded styles, as well as any combination of them. This offers a great flexibility when the styles are being created. Then the user can choose to mix the two types of rendering without losing the realistic illumination, as it can be seen in Figure 3.21a where the left style shades the bone in a realistic way and the right style highlight the teeth.

In addition to the usual properties, the editor allows the inclusion of illustrative effects. One of these effects is contour highlighting, introduced by (Bruckner and Groeller, 2007), where based on the normal projection on the sphere map, a different color can be assigned to the edge of the style in order to highlight contours in the rendered volume (Figure 3.21b). The color of the edge will be independent of the properties of the illumination, ensuring that even if the style is shaded the contours will be correctly rendered with the selected color.

Another effect, introduced by (Buchart et al., 2009), is the illustrative central transparency. This effect creates a smooth void in the center of the style, resulting in the material rendered with transparency in the zones that faces the viewer. The user can then see through other materials in order to have a better view of the key materials, without completely eliminating the materials in the middle (Figure 3.22a). This effect can be combined with

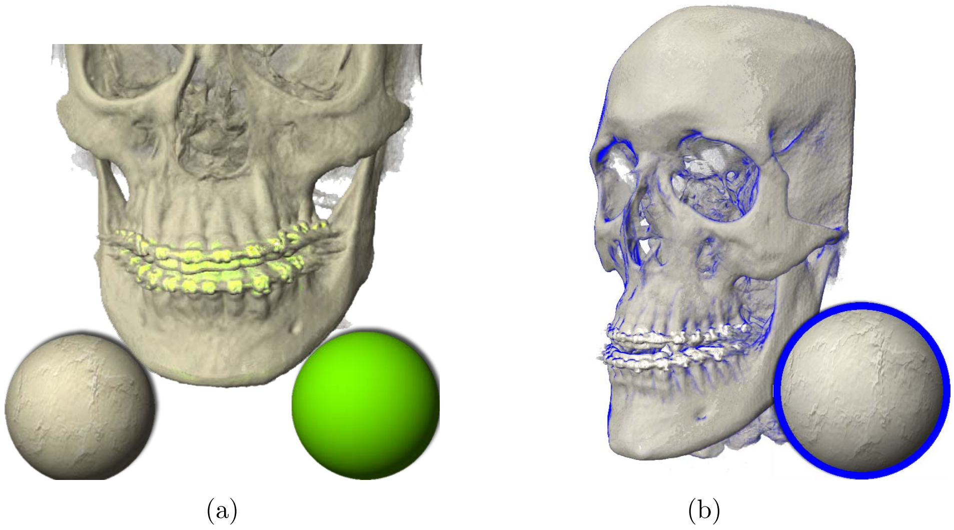


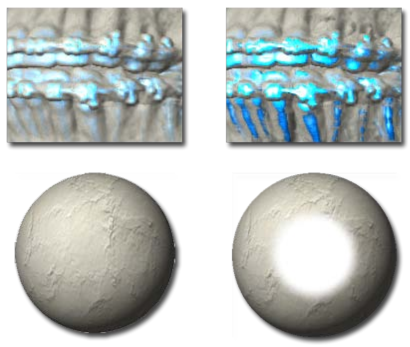
Figure 3.21: Images showing different effect types: Mixing of realistic and illustrative styles (a) and highlighting a skull using its contour (b).

the edge highlighting to create purely contour highlighting styles in order to see only the contour of a material (Figure 3.22b).

3.3.4 Extending Volumetric Visualization in VTK

In the previous sections has been explained that volume rendering is a highly variable field. Because of this variability, the development of a generic volumetric visualization library is a difficult task. Even if a library is developed, said variability forces the need of flexibility and personalization. Nevertheless, various libraries have been developed trying to fill this gap, with variable success.

One such library is the aforementioned Visualization Toolkit (VTK), which encompasses many fields and visualization modes including volume rendering. One key feature that has made VTK widely used is its modular approach and extensibility. Thanks to this, and to the fact that is an open source project, many modules are created by the users and later added to it.


(a) A style with the central transparency effect increases the visibility of the inner material and maintaining the outward material visible.

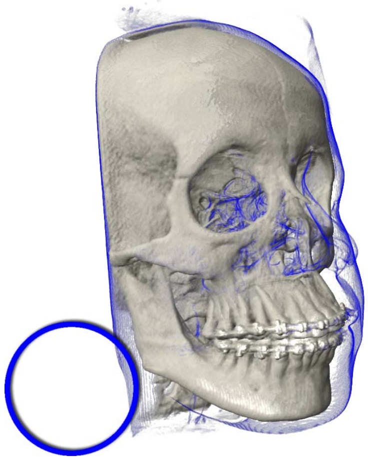

(b) Using an style which renders only contours, increases the perception of the context of the rest of the image by suggesting the contour of the surrounding material.

Figure 3.22: It is possible to add multiple effects to the styles and mix them using the proposed designer.

In addition to the standard rendering pipeline, VTK offers several volume rendering algorithms such as texture slicing or raycasting. However, the lack of the aforementioned improvements, such as stochastic jittering, shadow receiving volumes or direct volumetric isosurfaces, poses a problem in certain situations, such as a medical visualization using CTs. In order to fill this gap, in this dissertation a raycasting module for VTK has been developed and made available to the VTK community, allowing its use in any application already using VTK.

The first necessary addition to VTK for its use in surgery simulators is enabling the volumes to receive shadows. As it has been already explained, shadowing greatly increases the users ability to spatially locate the tool and its relation to the volume. It can be seen that this increases the potential learning done through a simulator as the interaction becomes much more natural.

VTK has the necessary modules to create a shadow mapping pipeline for polygonal objects. It does not, however, feature any volume rendering module able to cast or receive shadows. To enable shadow receiving for the volume rendering, the shadow mapping pipeline must feed the resulting depth map to the new volume rendering module. With this information, the new module is able to perform volumetric rendering with hard shadows. Moreover, thanks to the use of the native VTK pipeline for the creation of the shadow maps, the coherency between shadows received by polygons and volumes is ensured.

As it has been previously said the use of isosurfaces can be desirable given that some conditions are met. In the case of medicine, in some cases opaque tissues want to be visualized making the use of isosurfaces an evident choice. In VTK, there is no other option than the extraction of polygonal isosurfaces and their use for this type of rendering. For this reason, the presented new module features the use of direct volumetric isosurfaces.

Thanks to this added feature, there is no need to sacrifice visual quality for speed while continuing using the rest of VTK as before. Additionally, as it also has intersection refinement, the sampling distance can be varied to meet the necessary frame rate with nearly no visual quality loss. Additionally, in order to increase the quality of the FVR of the new module, stochastic jittering has been also added, greatly reducing the image quality.

These additions enable the use of VTK with the presented module for surgery simulation development, allowing developers to easily use it in their projects. It must be noted that shadow receiving is available in both FVR and isosurface rendering. To summarize, the additions done to VTK are these:

- The shadow receiving mode has been added, which is controlled in the same way it is controlled for polygonal objects seamlessly integrating with VTKs pipeline.
- A new rendering mode has been added, the volumetric isosurface mode. This mode also implements the intersection refinement allowing high quality renderings.
- Additionally, stochastic jittering has been added to further increase image quality.

Specific details about the changes in the class hierarchy and the modules themselves can be found in Appendix A.

3.4 Discussion

This chapter has presented the methods and algorithms necessary for rigid volumetric visualization in surgery simulation. Using these methods, the quality of the volumetric visualization can be increased with minimum impact on the frame rate. Additionally, some of them can be used to greatly increase the performance in certain cases.

In addition to presenting the methods, they have been implemented within VTK and published. This gives the developers easy access to these methods, which they can use without any specific knowledge of volumetric rendering.

Furthermore, an extension of the style transfer functions has been presented in the form of a style designer. This designer allows the user to easily create or modify style transfer functions to suit their needs. It can be used to create traditional transfer functions by uploading the image to be used and deactivating the illumination, to create realistic style transfer functions (with or without a texture) and to create illustrative style transfer functions using the different presented effects.

It should be noted that much more methods exist to improve a volumetric visualization, but they have not been explained as they fall outside the scope of this dissertation. That is the case of the methods for purely illustrative rendering methods, where much work has been done during the years, from cutaway and shading methods as in (Grau and Puig, 2009) or radically different rendering pipelines as in (Nagy et al., 2002) or (Ji et al., 2008).

Chapter 4

Volumetric Visualization of Deformable Models

Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.
Edsger W. Dijkstra

Part of this chapter has been published in:

Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Study of ray casting techniques for the visualization of deformable volumes". IEEE Transaction on Visualization and Computer Graphics, 2013. Accepted with Major Revision.

4.1 Introduction

Deformable body simulation plays a very important role in many areas, such as engineering and medicine. In some cases, the models to simulate are generated completely by computer, e.g., when a CAD model is tested by computer simulation prior to the development of a physical prototype. This method has the downside that the created model can contain differences or omission in comparison to the real model. And even in the case of considering a perfect model it cannot take into account individual differences, which is something to be taken into account in some cases.

In many other cases the models will simulate real objects, e.g. medical simulation, where the tissue to be simulated is usually extracted from volumes. One way to do so is by scanning the real object in order to generate a computerized version without having to manually create it. This way, using a very precise scanning a nearly perfect model will be achieved. And in the case where high quality scanning is not possible, as when scanning living beings, it allow to use models with individual differences.

These models need to be processed before being usable in a simulation pipeline. Usually this means the creation of a tetrahedral or hexahedral mesh from the original volume or point cloud. When using point clouds, the cloud is converted to a surface and then to a polyhedral volume by assuming uniformity inside the model. In contrast, when using volume data to create a polyhedral mesh, the information of the inside of the object no longer has to be guessed, as accurate information is provided. By using this information a more exact physical model can be created.

Usually, after the creation of the model the volume is no longer used and the visualization is performed with the model itself or by processing said model. But in the case of simulations performed in structures extracted from volumes, the volume's information can be added to the resulting shape to increase the quality of the visualization as well as the quantity of information given by it.

Volume rendering is required to perform this visualization, but most volume rendering pipelines do not allow the deformation of the volume. One of the most common volume rendering technique is raycasting, which does not allow the deformation of the volume without additional work. The common pipeline performs a raycasting in the original structured volume, requiring substantial changes in the algorithm to visualize the deformed model.

One option is to resample the volume using the deformation of the volume but this is a costly operation, both in terms of memory and time consumption, and therefore it is better to avoid it. The second option is the inverse deformation of the cast rays.

When inversely deforming the rays, two different spaces have to be considered: world space and volume space. The world space is the main space, in which all the objects in the scene are defined. Volume space is the space where the volume is sampled. This space is deformed (without deforming what is inside it), and the casted rays are deformed when transforming them from world space to volume space.

This deformation renders some assumptions of the usual raycasting pipeline invalid, such as the continuity of the rays or the static gradients of the volume. Therefore, additional corrections must be made in order to face these challenges. For example, as the volume itself is not modified, the gradients of the volume (usually used for illumination calculation) must be corrected to match the deformation. In the last decade several methods have been developed to perform the inverse deformation of the rays in interactive rendering times, allowing the rendering of deformable models. In the context of this dissertation, deformable volumetric rendering will be necessary for the second stage of the simulator, as it can be seen in Section 5.3.

This chapter will present the methods that have been developed to perform the inverse deformation of the rays in interactive rendering times as well as the challenges found in said development. Furthermore, different optimizations and additions will be presented. Finally, a comparison of the performance and visual result of the methods will be shown, providing an experimental background to those seeking to develop an application with such needs.

4.2 State of The Art of Deformable Volumetric Visualization

Many volume rendering methods for deformable models have been proposed through the years, creating two clearly different categories: non physics based and physics based deformation visualization.

The first category encompasses methods to visualize a deformed volume without having any precise physical background for the deformation. In this category a common technique is the deformation of rays, first proposed in (Kurzion and Yagel, 1997). Using this idea, Chen et al. proposed a deformation model specified by a Free Form Deformation (FFD) and visualized with a raycasting and inverse ray deformation (Chen et al., 2001; Chen et al., 2003). In this way the volume can be deformed using the FFD instead of relying on predefined deflectors, thus achieving a more general deformation scheme. Extending this idea, (Westermann and Rezk-Salama,

2001) presented a method where the deformation is not specified by a whole FFD but by arbitrary surfaces in the volume, which are then used to deform the volume using texture slicing.

More recently, a method able to render a deformed volume using inverse ray deformation was presented by (Correa et al., 2010), where the deformations mimic physical simulations by adding different constraints to the each tissue.

Another commonly used method in this category is the use of skeletons to deform the volume. In this direction, (Gagvani and Silver, 2001) presented a skeleton guided volume deformation scheme usable for volume animation. Continuing this work, (Singh and Silver, 2004) presented a method to interact with the volume by extending the method and adding the capability to select and manipulate different parts of the volume. More recently, a method to deform a volume maintaining the correctness of the anatomy was presented by(Rhee et al., 2011).

This type of deformation can be very useful in certain tasks such as illustrative visualization (Correa et al., 2007) or browsing volumetric data (McGuffin et al., 2003; Birkeland and Viola, 2009) as shown in Figure 4.1. On the other hand, for a simulation application a rough approximation of a physical deformation is not enough. For these cases physics based deformation visualization is needed to achieve the desired accuracy.

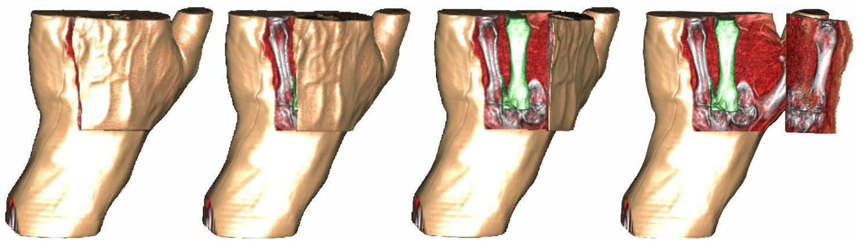


Figure 4.1: The sequence of a volumetric hand being peeled away in order to show the carpal bone of the middle finger (Image from (Birkeland and Viola, 2009)).

This second category groups the methods that include a physical simulation and those that, given their algorithms, can be used to visualize a simulated deformation. These methods are able to visualize a deformation from a physical simulation such as a finite element simulation or mass spring system with a underlying structured volume. One of the first methods was proposed by (Zhu et al., 1998) where they solve that problem by using the result of a finite element method (FEM) simulation to resample a deformed structured volume allowing the use of traditional volumetric rendering methods.

Although the resampling method allows any type of volume rendering, the computational cost of the resampling and the need to store two volumes (the original and the deformed) make it inefficient for interactive rendering or big structured volumes. Because of this, most methods developed are based on the idea of avoiding that operation. One early method able to do this was proposed by (Rezk-Salama et al., 2001) et al. where the deformation is achieved by deforming the texture space. Although this method is not directly usable with physical simulations it can be modified by adding an intermediate step to enable the use of simulation results, with the resulting overhead.

Another method is presented in the OpenGL Volumizer API (Bhanirantka and Demange, 2002; Jones and McGee, 2005), which renders a structured grid using a tetrahedral mesh as a proxy geometry usable with a FEM simulation. More recently, (Nakao et al., 2010) implemented a method simplifying this approach by using a single pass rendering. The main drawback of these methods is that they limit the rendering technique to be used to volume slicing, greatly restricting the possibilities of the visualization.

However, currently the most used volume rendering method is raycasting as it offers higher quality visualizations and a great flexibility. But before being able to use ray casting with an additional unstructured volume, an algorithm able to render a unstructured volume using raycasting is needed. The most used unstructured volumes are tetrahedral meshes, as they are the result of most used simulations, and are used to represent the state of a material in different aspects such as temperature or stress. As a result, most developed tetrahedral mesh rendering algorithms render the volume itself, without any underlying structured volume.

One of the first methods that uses raycasting to perform such rendering of tetrahedral meshes was presented by (Weiler et al., 2003). Continuing with this work, an improved algorithm was presented (Weiler et al., 2004), where the structures were optimized for the GPUs. Continuing with this work, an improved algorithm was presented (Weiler et al., 2004), where the structures were optimized for the GPUs. Later, an improved calculation of the entry faces using a depth peeling technique was presented by (Bernardon et al., 2006). In the same year, (Marmitt and Slusallek, 2006) presented a raycasting algorithm for the CPU with an improved ray tetrahedron intersection method using Plücker coordinates. More recently in (Muigg et al., 2007; Muigg et al., 2011) a method able to render more general grids, e.g. hexaedral grids, has been presented. These algorithms achieve high quality visualizations, as shown in Figure 4.2. However these methods only deal with unstructured volumes without taking into account any other volume.

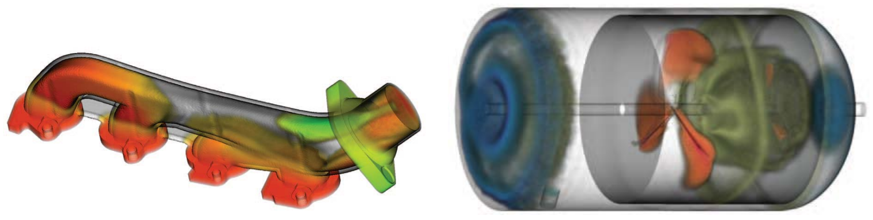


Figure 4.2: Raycasted images of some unstructured volumes resulting from computer simulations (image from (Muigg et al., 2011)).

In the last years, with the increase in the available computational power and the reduction of the price of these components, applications where real-time simulation and structured volumes are be mixed are becoming common. It can be seen that in order to visualize a structured volume with the deformation defined by a unstructured volume using raycasting both visualization modes have to be mixed. An algorithm to visualize unstructured grids with an underlying structured volume was first presented in (Tejada and Ertl, 2005) where the inside of the tetrahedra was rendered with pre integration. In that direction, (Georgii and Westermann, 2006) presented a raycasting method requiring the sorting of the tetrahedra, which they avoid by using a set of spherical shells.

However, these algorithms were molded by the restrictions of the hardware, which have rapidly evolved from that time. GPUs have greatly increased their features power and abilities over the years and by taking advantage of this improvements, a new raycasting algorithm for unstructured volumes with underlying volumes can be developed with higher capabilities. Additionally, these advances enable the addition of new features and the increase of the rendering quality.

4.3 Deformable Volumetric Visualization

One of the most flexible and promising methods to render a deformable model is to have the shape of the object and its appearance separated and rendering the deformable volume using inverse ray deformation.

An unstructured volume can be used to define the shape of the object, which changes with the physical simulation. At the same time, a structured volume will define the appearance of the object, in a similar way to a 2D texture defining the appearance of a polygonal surface. This can be seen as the definition of two different spaces: world space and volume space. The world space is the usual space in which the models in the scene are defined, including the unstructured mesh. The volume space, on the other hand is the space of the structured volume. The relation between these two spaces is defined by the unstructured mesh, as the tetrahedral mesh is the container of the volume space.

It should be noted that the rendering pipeline presented in this dissertation uses only tetrahedral volumes, as they are the simplest three-dimensional polyhedra. As a result, any other can be converted to a set of tetrahedra, making the tetrahedra a sensible choice for the base representation. Additionally, as the algorithm is executed in the GPU, the mesh must be uploaded using a data format suitable for its by shaders. Tetrahedra are easily defined only with their vertices, by implicitly storing the connection between them by using a consistent ordering to store them. Furthermore, by storing only one additional neighbouring tetrahedra information per tetrahedron, a full tetrahedral mesh is completely defined.

In order to store all this data in the GPU, three textures are used. The vertices texture contains all the vertices of the tetrahedral mesh, saving memory space as duplicated nodes are avoided. The tetrahedra texture stores the indices of the nodes that conform each tetrahedra in a specific order defining the four faces with the known combinations. The neighbours texture contains the index of the neighbouring tetrahedron for each face of each tetrahedron, once again in a known order to avoid additional information storage.

The initial change to the pipeline is the use of raycasting to render the tetrahedral mesh instead of the structured volume. Raycasting a tetrahedral volume requires a modified pipeline which usually resembles the flow chart shown in Figure 4.3.

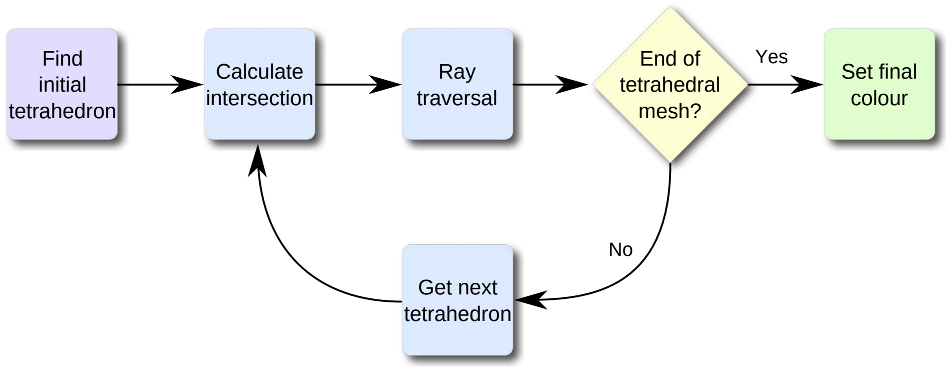


Figure 4.3: Flow chart illustrating the usual raycasting pipeline for tetrahedral meshes.

A common method to find the initial tetrahedron is to extract the surface of the tetrahedral mesh and then use spatial partitioning structures (such as kd-trees) to find the initial tetrahedron for every pixel. But given that this method is aimed to be executed directly in the GPU, it is highly desirable to avoid these kind of data structures.

Instead of finding the initial tetrahedron for each ray through a spatial partitioning structure, a method similar to the one presented in (Weiler et al., 2004) is used. Under this scheme, the surface tetrahedra are rendered as polygons encoded with the index of the tetrahedra. This completely avoids the use of additional data structures completely, reducing the overhead.

This method includes an additional step to allow the use of non convex meshes: a depth peeling technique is applied to the surface in order to extract the reentrant faces. With this information, when a final tetrahedron is found (one without neighbours in the exit face), instead of ending the traversal, the next reentrant face for the ray is fetched. In addition to calculating the entry point of the rays into the tetrahedra, this method directly skips the empty spaces inside the unstructured volume. This step has been simplified in the presented algorithm by setting a maximum number of reentrant layers in order to reduce its computational cost and avoiding the need for multiple volume rendering passes. It has been

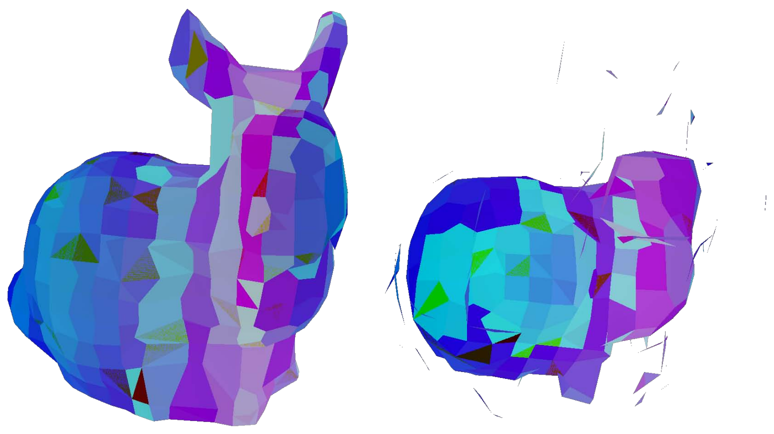


Figure 4.4: The first and second layers of entry tetrahedra of the Stanford Bunny. As the bunny is empty inside, the ray exits from the tetrahedral mesh in the gap inside it and enters the tetrahedron in the second layer to continue the ray traversal. It should be noted that although some faces may seem to share a colour due to the encoding, each face has a unique code.

experimentally found that some of the most common types of meshes need as few as three layers. However, in order to take into account possible variations in this number, five layers have been used in the performed tests. It should be noted, though, that the overhead introduced by increasing the number of layers affects mostly to the needed memory, as the impact in performance is nearly nonexistent.

The usual method to perform the depth peeling discards the back faces of the surface and then compares the depth of the different layers, obtaining the different entry layers. In most cases this method is perfectly valid, as the orientation of the tetrahedra, and therefore of the faces, is known. However, in the event of not knowing the orientation or it being arbitrary, a modified version must be used. For this purpose, instead of not rendering the back faces, the faces can be rendered regardless of orientation with the depth peeling. If there are no self intersections in the tetrahedral mesh, it can be known that the odd layers of the depth peeling will be the entry faces, and the even will correspond to the exit faces. Then, by storing the odd layers the result will be equal to the first method.

Figure 4.4 shows an example of the different entry layers, where each tetrahedron is coded by a different colour for illustrative purposes. Since this colour coding introduces rounding errors when retrieving the original index (due to the floating point precision), the algorithm avoids it by directly using the element's index. This is made possible by using frame buffer objects, as they allow to render to a texture without the limitation of each pixel to be between 0 and 1. This enables the direct use of the index number as the colour when rendering, removing the rounding errors.

Once the first tetrahedron is found, the intersection of the ray with this tetrahedron is calculated and the exit face is found. Combined with the stored neighbouring information, the next tetrahedron can be fetched without any additional calculations. Once a tetrahedron without a neighbour in the exit face is found, the next layer of entry faces is fetched and the process continues.

In most algorithms, the traditional ray tetrahedron intersection method of (Haines, 1991) is used , as in (Tejada and Ertl, 2005). In the method presented in this dissertation, however, Plücker coordinates are used to calculate the ray tetrahedron intersection using a version of the algorithm presented (Platis and Theoharis, 2003), modified for its efficient use in the GPU.

Plücker coordinates define a ray in three-dimensional space as shown in Equation 4.1. As it can be seen, Plücker coordinates assign a six dimensional representation for the lines.

$$\pi_r = \{L : L \times P\} = \{U_r : V_r\} \tag{4.1}$$

Being P a point in the ray, L its direction and π_r the resulting ray representation. Plücker coordinates have their own properties that are not going to be discussed in this dissertation, for further information please consult (Stolfi, 1987). Instead, the most interesting property of the Plücker coordinates will be presented, an operation called permuted inner product (Equation 4.2).

$$\pi_r \odot \pi_s = U_r \cdot V_s + U_s \cdot V_r \tag{4.2}$$

With the resulting scalar of the permuted inner product, the orientation of s relative to r can be known using Equation 4.3.

$$\begin{cases} \pi_r \odot \pi_s > 0 & \text{Ray } s \text{ goes counter clockwise around } r \\ \pi_r \odot \pi_s < 0 & \text{Ray } s \text{ goes clockwise around } r \\ \pi_r \odot \pi_s = 0 & s \text{ and } r \text{ are coplanar} \end{cases} \tag{4.3}$$

This information can be used to calculate the intersection of a ray with the faces of a tetrahedron by performing the permuted inner product of the ray with each of the segments of the face. If the result of these calculations have the same sign, the ray intersects with the face, otherwise the ray does not intersect. Furthermore, if the orientation of the faces is controlled, it can be known if the ray enters or exits the tetrahedron by checking the sign of the results, as shown in Figure 4.5. If all the values are positive the ray enters the element, while, if they are all negative the ray exits it (assuming outwards oriented tetrahedra). Finally, if they are all zero the ray is coplanar to the face. If none of this criteria is met, the ray does not intersect the current face.

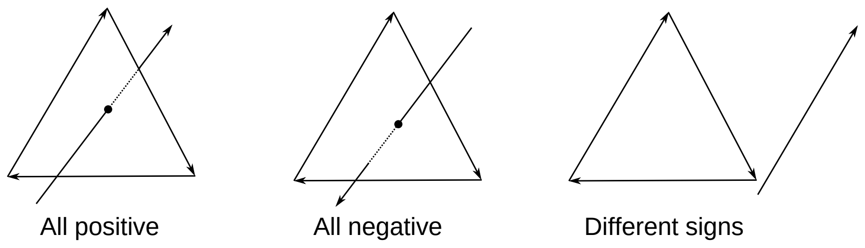


Figure 4.5: The ray triangle intersection performed with Plücker coordinates. Note that the case of coplanar rays is not shown in this figure.

This method reduces the number of operations required to calculate the intersection of the ray. Additionally, when the ray intersects a face the permuted inner product of the ray with the different edges directly provides the unscaled barycentric coordinates of the intersection point, avoiding the extra calculations to calculate them. The barycentric coordinates are computed as $u_i^k = w_i^k / \sum_{i=0}^{3} w_i^k$ being $w_i^k = \pi_r \odot \pi_{e_i}$ and e_i the Plücker coordinates for each edge of the face.

Summarizing, the ray tetrahedron intersection method using Plücker coordinates starts by calculating the intersection of the ray with the first tetrahedron. For this purpose, the intersection is calculated with each face so that the entry and exit face are known. This needs the checking of two conditions per face, to check whether they are all negative (exit) or all positive (entry), as it can be seen in Algorithm 4.1.

Algorithm 4.1 Base Plücker ray tetrahedron intersection test.

```
1: function Ray-Tetrahedron Intersection($Ray$, $TetraEdges$, $InPoint$, $InFace$, $OutPoint$, $OutFace$)
2:     for $i = 0 \to 4$ do
3:         $\pi_0 = Ray \odot TetraEdges[i][0]$
4:         $\pi_1 = Ray \odot TetraEdges[i][1]$
5:         $\pi_2 = Ray \odot TetraEdges[i][2]$
6:         if $\pi_0 >= 0 \wedge \pi_1 >= 0 \wedge \pi_2 >= 0$ then
7:             $InPoint = ConvertToBarycentric(\pi_0, \pi_1, \pi_2)$
8:             $InFace = i$
9:         else if $\pi_0 < 0 \wedge \pi_1 < 0 \wedge \pi_2 < 0$ then
10:            $OutPoint = ConvertToBarycentric(\pi_0, \pi_1, \pi_2)$
11:            $OutFace = i$
12:        end if
13:    end for
14: end function
```

Knowing that the ray must intersect with one of the faces and that the exit point of the current tetrahedron will be the entry point of the next, in the rest of intersections only the exit point needs to be calculated, removing the entry face condition, although it should be noted that the condition to be met is comprised of three comparisons, one for each edge of the face. In addition, as only one checking has to be done the loop that calculates the intersection with the faces can stop when the desired point has been found without extra condition checking. Furthermore, the reusing of the exit point ensures that there are no incoherences between the exit point of the current tetrahedron and the entry point of the next, something that happens when both points computed due to numerical errors.

However, as one ray per pixel is used, it can be assumed that rays will intersect with the whole tetrahedra and that rays passing the edges of the faces will have to be calculated. With the aforementioned method, in these cases numerical errors can cause not to find the entry or exit point, causing pixel sized artifacts as shown in Figure 4.6. These artifacts are most noticeable in moving models or with a moving camera as they rapidly flicker.

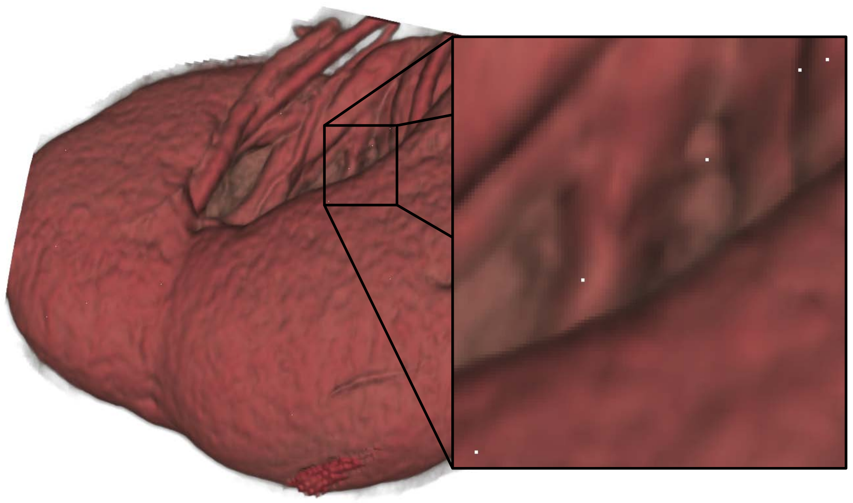


Figure 4.6: Visualization of a kidney, showing the visual artifacts created by the numerical errors in the intersection calculation.

But, as it is known that rays will always intersect with the tetrahedron, the algorithm can be modified to avoid these artifacts. This modification must ensure that an entry and exit point are calculated, regardless of meeting the exact conditions or not. To do so, a method to determine how close each set of values is to meet the conditions must be used and then select the face closest to meeting those conditions.

In the case of the entry point, π_0, π_1 and π_2 are taken and clamped to be in $(-\infty, 0.0]$. By doing so, the positive values are clamped to zero while the negative are maintained. By adding the three numbers, a numeric value of how close that face is to meet the entry face condition (the three values are positive) is achieved. By taking the face with this value closest to zero the entry point can be calculated. The same procedure is used with the exit point, clamping to $(0.0, \infty]$.

With this method, when the ray does not incur in any numerical errors, the correct faces are selected. And when numerical problems arise, the faces closest to meeting the conditions are selected, effectively fixing the visual artifacts. It should be noted though, that when the entry and exit points are very close together, this method may occasionally give them inverted (the entry point as exit point). Instead of clamping to zero, a small error magniturde (ϵ) can be used: clamping to $(-\infty, \epsilon]$ for the entry face and $(-\epsilon, \infty]$ for the exit face, avoiding the inversion problem. It has been empirically found that ϵ can be left aside in most cases, and that in the cases where it was required values between 0.1 and 1 gave equally good results.

Algorithm 4.2 Plücker ray tetrahedron intersection test modified to avoid visual artifacts.

```
1: function IMPROVED RAY-TETRAHEDRON INTERSECTION($Ray$, $TetraEdges$, $InPoint$, $InFace$, $OutPoint$, $OutFace$)
2:     $MaxValue = -\infty$
3:     $MinValue = \infty$
4:     for $i = 0 \to 4$ do
5:         $\pi_0 = Ray \odot TetraEdges[i][0]$
6:         $\pi_1 = Ray \odot TetraEdges[i][1]$
7:         $\pi_2 = Ray \odot TetraEdges[i][2]$
8:         $Sum = Min(\pi_0, \epsilon) + Min(\pi_1, \epsilon) + Min(\pi_2, \epsilon)$
9:         if $Sum > MaxValue$ then
10:            $MaxValue = Sum$
11:            $InPoint = ConvertToBarycentric(\pi_0, \pi_1, \pi_2)$
12:            $InFace = i$
13:        end if
14:        $Sum = Max(\pi_0, -\epsilon) + Max(\pi_1, -\epsilon) + Max(\pi_2, -\epsilon)$
15:        if $Sum < MinValue$ then
16:            $MinValue = Sum$
17:            $OutPoint = ConvertToBarycentric(\pi_0, \pi_1, \pi_2)$
18:            $OutFace = i$
19:        end if
20:    end for
21: end function
```

Algorithm 4.2 shows the modified method to calculate the intersection. It can be seen that it requires more conditions than the previous, as each $min()$ and $max()$ require an additional comparison, needing a total of four comparisons per face. Additionally, some calculations are now performed per each face instead of calculating it only for the correct one. However, the added cost is nearly unnoticeable in the performed tests as most GPUs perform better by calculating a value more than once than performing a comparison to check whether it should be calculated or not. Furthermore, as with the previous method, once the entry and exit points for the first tetrahedra have been found, the rest of the intersections only have to calculate the exit point.

Algorithm 4.3 Plücker ray tetrahedron intersection test modified to include orientation checking.

```
function IMPROVED RAY-TETRAHEDRON INTERSECTION($Ray$, $Vtx$, $TetraEdges$, $InPoint$, $InFace$, $OutPoint$, $OutFace$)
    ...
    $Or = -TripleProduct(Vtx[1]-Vtx[0], Vtx[2]-Vtx[0], Vtx[3]-Vtx[0])$
    for $i = 0 \to 4$ do
        $\pi_0 = (Ray \odot TetraEdges[i][0]) * Or$
        $\pi_1 = (Ray \odot TetraEdges[i][1]) * Or$
        $\pi_2 = (Ray \odot TetraEdges[i][2]) * Or$
        ...
    end for
end function
```

Another consideration is that, in a similar manner to the finding of the initial tetrahedron, this method is dependant of the orientation of the tetrahedra. If the orientation is controlled, it is known which comparison gives the entry point and which gives the exit point. If the orientation is not is arbitrary or unknown, an additional control is needed. The orientation of the tetrahedra can be easily calculated using the scalar triple product. The resulting scalar will be negative if the faces are oriented outwards whereas if the faces are oriented inwards it will be positive (if it is zero the tetrahedron is collapsed onto a single plane). The additional comparison can be avoided by multiplying the π_i with the negative of the triple scalar, avoiding the need to further change the aforementioned algorithm, as shown in Algorithm 4.3 where Or is the result of the triple product while Vtx are vertices of the tetrahedron being checked.

However, in most tetrahedral meshes the orientation is known or it can be controlled in the creation process. Due to this, most cases that may need to check for inverted tetrahedra arise from the deformation of the mesh by a simulation. Additionally, the inversion of a tetrahedron also renders the assumption of no self intersections within the tetrahedral mesh invalid. This requires changing the initial face finding, because the orientation and the order of the faces are no longer useful, the ray traversal in the tetrahedral mesh as the same ray segment can traverse more than one tetrahedra at the same time, and the next tetrahedron finding as there may not be a single exit face.

Nevertheless, this case is rather uncommon as the inversion of tetrahedra in physical simulations is usually unwanted and can lead to a failure in the simulation. Taking this into account, Algorithm 4.2 will be used in the rest of the dissertation and the usual depth peeling, without correction for arbitrary orientation, for the initial faces. This way, the presented results will be more useful for the development of these methods in surgery simulation applications.

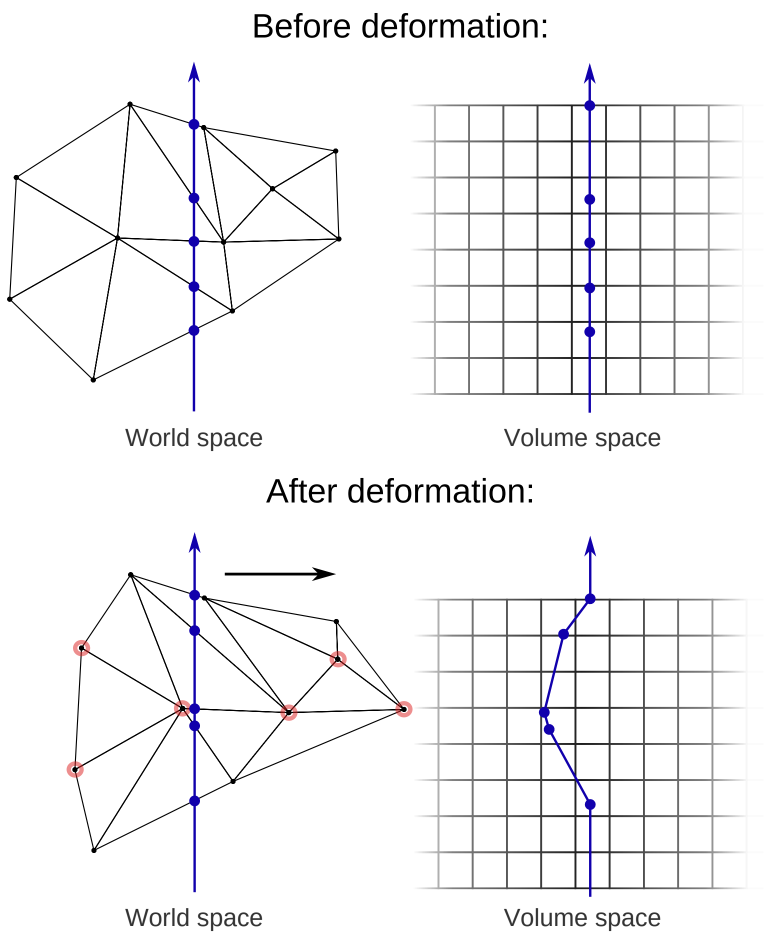


Figure 4.7: The triangular model is shifted rightwards and the ray (in volume space) is deformed leftwards to match the deformation.

All the calculations up to this point have been performed in world space, as the rays and the tetrahedra are defined in it. Now, in order to perform the ray traversal in volume space, the ray segments must be transformed from world space to volume space. For this purpose, the barycentric coordinates of the entry and exit points in their respective faces current tetrahedron are calculated. Using these coordinates, a barycentric interpolation of the structured volume coordinates of the nodes is performed, as shown in Equation 4.4. To do so, the position in volume coordinates (in the structured volume) of each vertex of the tetrahedral mesh in the initial position is stored beforehand.

This coordinates dictate the initial and ending point of the current ray segment in volume space, effectively transforming the ray segment from world space to volume space. In volume space the ray segment will have sustained a deformation inverse to the deformation of the tetrahedral mesh, achieving the same effect as deforming and resampling the structured volume. Figure 4.7 shows a two-dimensional example of this deformation, where the left side is the world space and the right side is the volume space. It can be seen that for the undeformed triangular mesh the two spaces are equal, but when the mesh is deformed the ray is inversely deformed in volume space.

In addition to deform the ray, by interpolating in the same way the world position of the vertices, the positions of the ray in world are easily calculated. With this information, the depth information necessary for the calculation of the shadows, the intersection with polygonal objects or for a correct depth calculation is easily obtained. For example, in the case of the entry point this is the data available: b_0, b_2, b_3 are the barycentric coordinates of the entry point respect to the entry face, $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}$ are the volume coordinates of the nodes of the entry face and $\mathbf{w_0}, \mathbf{w_1}, \mathbf{w_2}$ are the world coordinates of the nodes of the entry face. With this data, we can calculate both the entry point in volume coordinates ($\mathbf{EntryPoint_{Volume}}$) and world coordinates ($\mathbf{EntryPoint_{World}}$) with:

$$\begin{aligned}\mathbf{EntryPoint_{Volume}} &= b_0 * \mathbf{v_0} + b_1 * \mathbf{v_1} + b_2 + \mathbf{v_2} \\ \mathbf{EntryPoint_{World}} &= b_0 * \mathbf{w_0} + b_1 * \mathbf{w_1} + b_2 + \mathbf{w_2}\end{aligned} \tag{4.4}$$

Summarizing, this method allows the use of a tetrahedral mesh to deform the visualization of a structured volume. This algorithm allows maintaining the quality of raycasting and the use of methods commonly

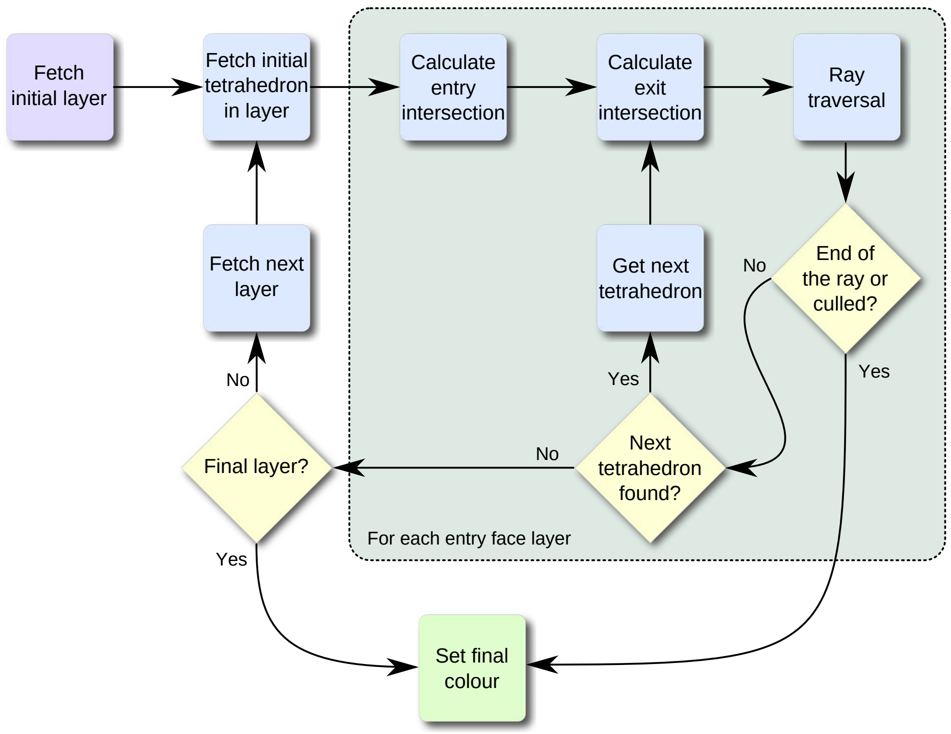

Fetch initial layer
Fetch initial tetrahedron in layer
Calculate entry intersection
Calculate exit intersection
Ray traversal
Fetch next layer
Get next tetrahedron
No
End of the ray or culled?
Yes
No
Yes
Final layer?
No
Next tetrahedron found?
For each entry face layer
Yes
Set final colour

Figure 4.8: Flow chart illustrating the proposed raycasting pipeline for tetrahedral meshes with underlying structured volumes.

used with minor modifications, and avoid resampling the structured volume. As a result, thanks to this method, the ray traversal inside each ray segment can be calculated as explained in Chapter 3, and the presented improvement methods can also be used. In order to give an overview of the algorithm, a flow chart of the final improved algorithm is shown in Figure 4.8.

As the ray traversal is performed in volume space, where the ray is deformed, some assumptions made in the traditional raycasting pipelines are invalid, such as the linearity of the ray. As a result, additional steps must be added to the raycasting pipeline to achieve high quality renderings, as explained in the following sections.

4.4 Pipeline improvements

4.4.1 Ray traversal

One of the differences this method presents, is that as it can be seen in Figure 4.7, the ray is transformed from a single line into a series of connected segments with C^0 continuity. But when sampling the segments an additional problem arises, as additional errors can appear when the given sample distance does not exactly reach the ending point. There are three main options to deal with this:

1. Traverse the entire ray from the first to the last point of the ray. In this case, the traversal is continued until the next sample is outside the ray. This method adds additional samples at the end of each ray, which causes visual artifacts in certain cases making the tetrahedra very noticeable.

   These artifacts can be mitigated by moving the first sample of the next ray with the distance left from the previous ray to complete a step. Additionally, the opacity of the first and last samples of the ray are corrected to match the used distance. Although these methods greatly improve the quality, some artifacts continue being visible. We have called this ***no rounding***.

2. Start the ray segment on the last sampling point of the previous tetrahedra. This way, no additional calculations have to be performed and no additional samples are taken, completely avoiding the visual artifacts.

   This method creates a deviation from the original ray, but this deviation does not create any artifacts. We have called this ***ceil rounding***.

3. Traverse the ray inside the tetrahedra from the entrance point to the exit point, but stopping if the distance to the exit point is smaller than the step size. We have called this ***floor rounding***.

   Instead of sampling the final point of the ray and correcting the opacity, in this method the last point is not sampled if the distance requirement is not met. Then, the first sample of the next ray is moved so that the sum of the distance left in the previous ray and the distance to the first sample is the desired sample distance. Thanks to this, no artifacts are visible using this method.

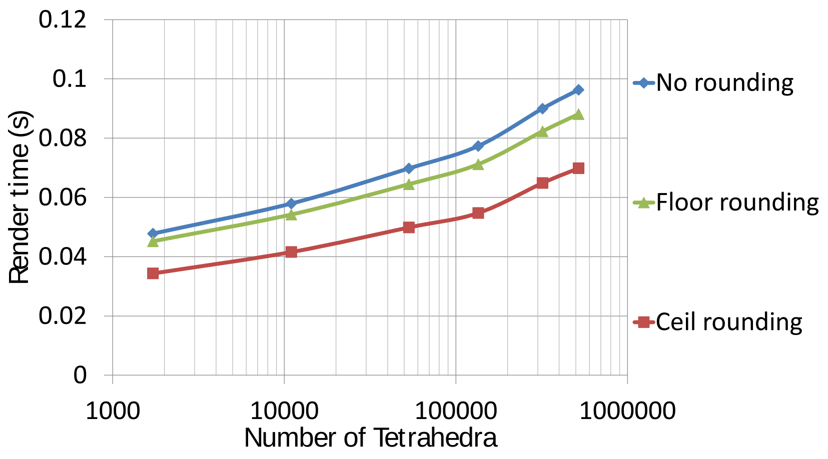


Figure 4.9: Comparison between the render times of the three options: ***no rounding***, ***ceil rounding*** and ***floor rounding***.

Within these methods, ***ceil rounding*** and ***floor rounding*** options give the best visual results with no differences between them. However, as it can be seen in Figure 4.9, ***floor rounding*** is significantly slower than ***ceil rounding*** making it a better option. It should be noted that the ***no rounding*** option is the slowest of the three, because of the need of additional corrections.

4.4.2 G^1 continuity

In some cases additional continuity may be desirable, as the current method only ensures C^0 continuity. Theoretically this could allow the smoothing of the final visualization, resulting in less noticeable artifacts due to the underlying tetrahedral mesh.

Cubic Hermite splines offer a good choice in this regard as only two points and two tangents are needed to define them: the entry ($\mathbf{p_0}$) and exit ($\mathbf{p_1}$) points. To calculate the tangents of these points ($\mathbf{m_0}$ and $\mathbf{m_1}$ respectively) the previous starting point and the next starting point are also needed. Equation 4.5 defines cubic Hermite spline.

$$p\left(t\right)=b_{00}\mathbf{p_0}+b_{01}\mathbf{m_0}+b_{10}\mathbf{p_1}+b_{11}\mathbf{m_1} \tag{4.5}$$

where:

$$\begin{aligned}
b_{00}&=2t^3-3t^2+1\\
b_{01}&=t^3-2t^2+t\\
b_{10}&=-2t^3+3t^2\\
b_{11}&=t^3-t^2
\end{aligned} \tag{4.6}$$

Additionally, it has been found that the quality of the image rendered with smoothed rays can be improved by modulating the tangents' length according to the ray segment's length. This converts the C^1 continuity into G^1 continuity, but the results improve and the possibilities of errors decrease dramatically as the speed of the spline is modulated according to its length.

Although using Hermite splines ensures G^1 continuity, it adds an additional problem when sampling the ray. The problem appears when calculating the desired step (Δt) in curve coordinates in order to use the desired sample distance.

If the correct distance in the curve space is desired the arc length has to be calculated by integrating the curve, as shown in Equation 4.7 where d is the objective sampling distance.

$$\int\limits_{t}^{t+\Delta t} p(t)\mathrm{d}t=d \tag{4.7}$$

However, in addition to being computationally intensive, this distance will not give an equal sampling distance in volume coordinates. If instead of calculating the correct distance in curve space it is calculated in volume space, an desired sampling distance is achieved while being less computationally intensive. Using this option, the next point in the curve at the desired sample distance in volume coordinates from the current point can be found. In this sense, Equation 4.8 must be fulfilled.

$$|p(t+\Delta t)-p(t)|=d \tag{4.8}$$

By taking only the $p(t+\Delta t) - p(t)$ part:

$$\begin{aligned}
&\mathbf{p_0}\left(\left(2\left(t+\Delta t\right)^3 - 3\left(t+\Delta t\right)^2 + 1\right) - \left(2t^3 - 3t^2 + 1\right)\right) + \\
&\mathbf{m_0}\left(\left(\left(t+\Delta t\right)^3 - 2\left(t+\Delta t\right)^2 + \left(t+\Delta t\right)\right) - \left(t^3 - 2t^2 + t\right)\right) + \\
&\mathbf{p_1}\left(\left(-2\left(t+\Delta t\right)^3 + 3\left(t+\Delta t\right)^2\right) - \left(-2t^3 + 3t^2\right)\right) + \\
&\mathbf{m_1}\left(\left(\left(t+\Delta t\right)^3 - \left(t+\Delta t\right)^2\right) - \left(t^3 - t^2\right)\right)
\end{aligned} \tag{4.9}$$

Developing and simplifying these terms leaves Equation 4.10.

$$p\left(t+\Delta t\right) - p\left(t\right) = g_0\mathbf{p_0} + g_1\mathbf{m_0} + g_2\mathbf{p_1} + g_3\mathbf{m_1} \tag{4.10}$$

where:

$$\begin{aligned}
g_0 &= 2\Delta t^3 + \Delta t^2 \overbrace{\left(6t-3\right)}^{\alpha_1} + \Delta t \overbrace{\left(6t^2 - 6t\right)}^{\beta_1} \\
g_1 &= \Delta t^3 + \Delta t^2 \overbrace{\left(3t-2\right)}^{\alpha_2} + \Delta t \overbrace{\left(3t^2 - 4t + 1\right)}^{\beta_2} \\
g_2 &= -2\Delta t^3 + \Delta t^2 \overbrace{\left(-6t+3\right)}^{\alpha_3} + \Delta t \overbrace{\left(-6t^2 + 6t\right)}^{\beta_3} \\
g_3 &= \Delta t^3 + \Delta t^2 \overbrace{\left(3t-1\right)}^{\alpha_4} + \Delta t \overbrace{\left(3t^2 - 2t\right)}^{\beta_4}
\end{aligned} \tag{4.11}$$

By grouping and rearranging Equation 4.10 the following result is achieved. Note that $\mathbf{k_3}$ is equal to the first derivative of the cubic Hermite spline (Eq. 4.5).

$$\begin{aligned}
&p\left(t+\Delta t\right) - p\left(t\right) = \Delta t^3\mathbf{k_1} + \Delta t^2\mathbf{k_2} + \Delta t\mathbf{k_3} \\
&\mathbf{k_1} = 2\mathbf{p_0} + \mathbf{m_0} - 2\mathbf{p_1} + \mathbf{m_1} \\
&\mathbf{k_2} = \alpha_1\mathbf{p_0} + \alpha_2\mathbf{m_0} + \alpha_3\mathbf{p_1} + \alpha_4\mathbf{m_1} \\
&\mathbf{k_3} = \beta_1\mathbf{p_0} + \beta_2\mathbf{m_0} + \beta_3\mathbf{p_1} + \beta_4\mathbf{m_1}
\end{aligned} \tag{4.12}$$

Then, continuing with Equation 4.8:

$$|p(t+\Delta t) - p(t)|^2 = \sum_{i=x,y,z} \left(\Delta t^3 \mathbf{k_{1_i}} + \Delta t^2 \mathbf{k_{2_i}} + \Delta t \mathbf{k_{3_i}}\right)^2 \tag{4.13}$$

Expanding this expression:

$$\sum_{i=x,y,z} \mathbf{k_{1_i}}^2 \Delta t^6 + 2\mathbf{k_{1_i}}\mathbf{k_{2_i}}\Delta t^5 + \left(2\mathbf{k_{1_i}}\mathbf{k_{3_i}} + \mathbf{k_{2_i}}^2\right)\Delta t^4 + 2\mathbf{k_{2_i}}\mathbf{k_{3_i}}\Delta t^3 + \mathbf{k_{3_i}}^2 \Delta t^2 \tag{4.14}$$

Rearranging and grouping these terms leaves Equation 4.15.

$$|p(t+\Delta t) - p(t)|^2 = c_1 \Delta t^6 + c_2 \Delta t^5 + c_3 \Delta t^4 + c_4 \Delta t^3 + c_5 \Delta t^2 \tag{4.15}$$

where:

$$\begin{aligned} c_1 &= \mathbf{k_{1_x}}^2 + \mathbf{k_{1_y}}^2 + \mathbf{k_{1_z}}^2 \\ c_2 &= 2\mathbf{k_{1_x}}\mathbf{k_{2_x}} + 2\mathbf{k_{1_y}}\mathbf{k_{2_y}} + 2\mathbf{k_{1_z}}\mathbf{k_{2_z}} \\ c_3 &= 2\mathbf{k_{1_x}}\mathbf{k_{3_x}} + 2\mathbf{k_{1_y}}\mathbf{k_{3_y}} + 2\mathbf{k_{1_z}}\mathbf{k_{3_z}} + \mathbf{k_{2_x}}^2 + \mathbf{k_{2_y}}^2 + \mathbf{k_{2_z}}^2 \\ c_4 &= 2\mathbf{k_{2_x}}\mathbf{k_{3_x}} + 2\mathbf{k_{2_y}}\mathbf{k_{3_y}} + 2\mathbf{k_{2_z}}\mathbf{k_{3_z}} \\ c_5 &= \mathbf{k_{3_x}}^2 + \mathbf{k_{3_y}}^2 + \mathbf{k_{3_z}}^2 \end{aligned} \tag{4.16}$$

As it can be seen in Equation 4.15, the result is a sextic equation and, therefore, does not have an analytical solution. However, an approximation can be calculated by taking into account only up to the quadratic terms. This way an easily solvable equation is reached which can be used to calculate an approximation to the solution. Note that $\sqrt{c_5}$ is the length of the first derivative of the function, $p'(t)$.

It follows that $d^2 \approx \Delta t^2 c_5$, then $d^2 \approx \Delta t^2 |p'(t)|^2$ and it resolves with $\Delta t \approx d / |p'(t)|$ by taking the positive root. This linear approximation (called Δt_0) can be improved by iterating through Equation 4.15 with the Newton's method to get an improved approximation (Δt_n). As Newton's method may not converge depending on the starting point two additional constraints, shown in Equation 4.17, must be used to avoid overshooting.

$$\Delta t = \begin{cases} \Delta t_n & \text{if } 0 < \Delta t_n \leq 1 \\ \Delta t_0 & \text{otherwise} \end{cases} \tag{4.17}$$

During numerical tests, the refined approximation with just two iterations gave better results than the linear approximation, as it can be seen in Figure 4.10. It is important to take the dataset size into account when interpreting these results. For example, a volume with a size of 256^3 means that in volume coordinates a single voxel's length is 3.9×10^{-3} . This means that the Δt_0 error is barely within the same order of magnitude while Δt_n is one order more accurate.

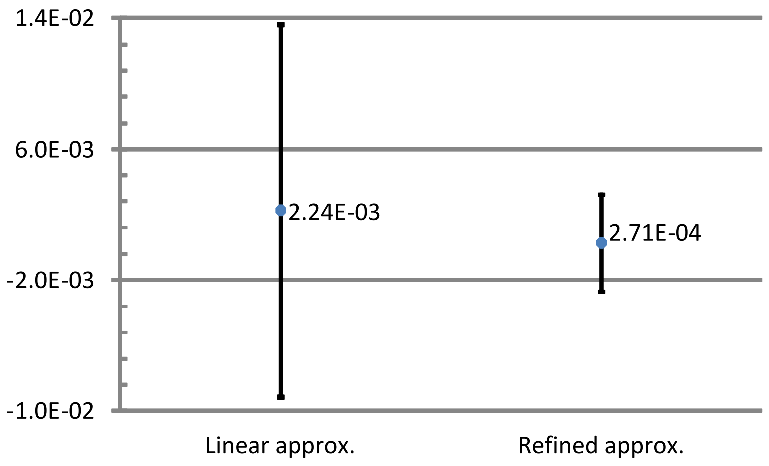


Figure 4.10: A comparison of the error using the linear approximation (left) and the refined approximation with two iterations (right).

Figure 4.11 shows the differences in the consistency of the sampling distances observable in the woodgrain artifacts, showing the linear approximation on the left and the improved on the right. The woodgrain artifact may be mitigated using stochastic jittering but this it has been intentionally left to better illustrate the sampling differences. It can be seen that an important improvement can be achieved with this method, without nearly any addition to the final computational cost. For this reason, when smoothed rays are used in the rest of the study, they will always be using the refinement with two iterations.

The increase on the level of continuity slightly improves the visualization quality, but before making a decision about the desired level of

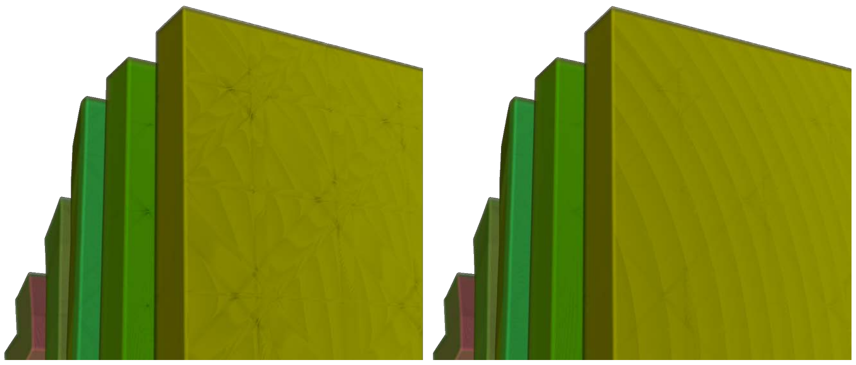


Figure 4.11: Comparison between G^1 continuity preserved using just a linear approximation (left), or using the refined result (right) with two iterations. The woodgrain artifacts gives visual clues of the sampling uniformity.

continuity to use, the rendering type and models to render have to be taken into account. As it can be seen in Figure 4.12, the difference can be seen when large deformations have taken place, and more notoriously around the edges. Additionally, this difference is more noticeable when isosurfaces are used, as FVR usually smooths the edges.

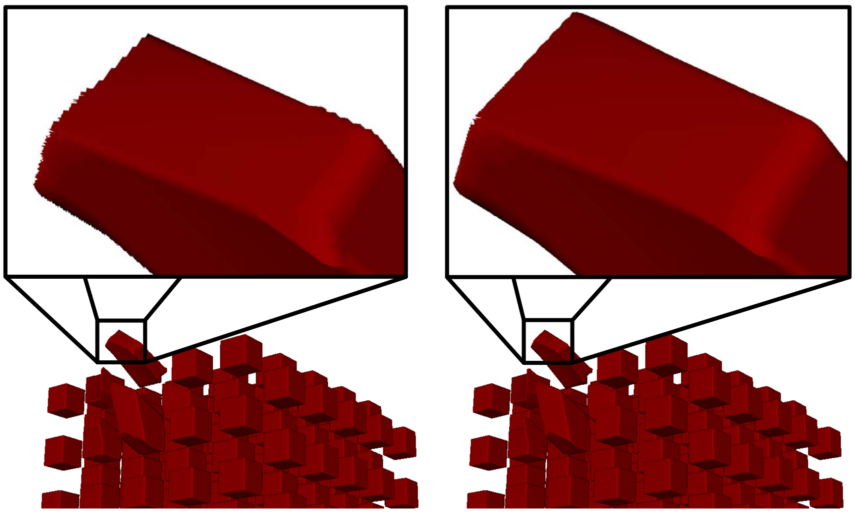


Figure 4.12: An isosurface rendering, showing the differences between C^0 (left) and G^1 continuity (right).

In addition to the quality, the rendering cost has to be taken into account. The use of Hermite splines implies the calculation of Δt and the corresponding point in the curve per sample, increasing the computational cost of the algorithm. As it is shown in Figure 4.13, the impact of ray smoothing varies depending on the chosen integration method, but it is a sizeable increase of computational cost. In the tests performed, isosurfaces show a more stable overhead than FVR, mainly given by the fact that in isosurfaces fewer tetrahedra are traversed. Section 4.5 gives expanded information on the performance of the smooth rays and the interaction with other methods.

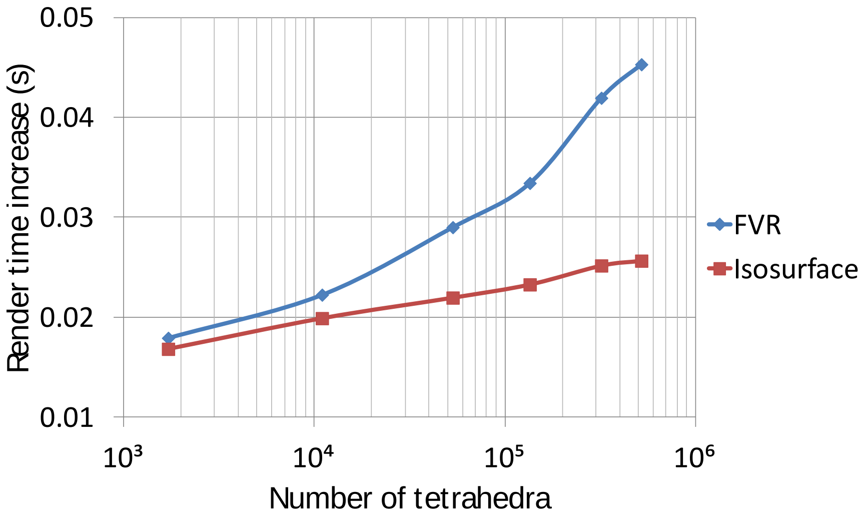


Figure 4.13: This chart shows the increase in rendering time when using G^1 continuity. The number of samples is constant among models (horizontal axis).

4.4.3 Illumination correction

The illumination of each sample is usually calculated using lighting models such as Phong or Goraud. The normal procedure in volume rendering is to approximate the normal with the gradient at the sample, but in the case of deformable rendering the structured volume's space is different to the world space so it cannot be directly used for this calculation: it must be transformed from the structured volume coordinates to world coordinates, as it can be seen in Figure 4.14.

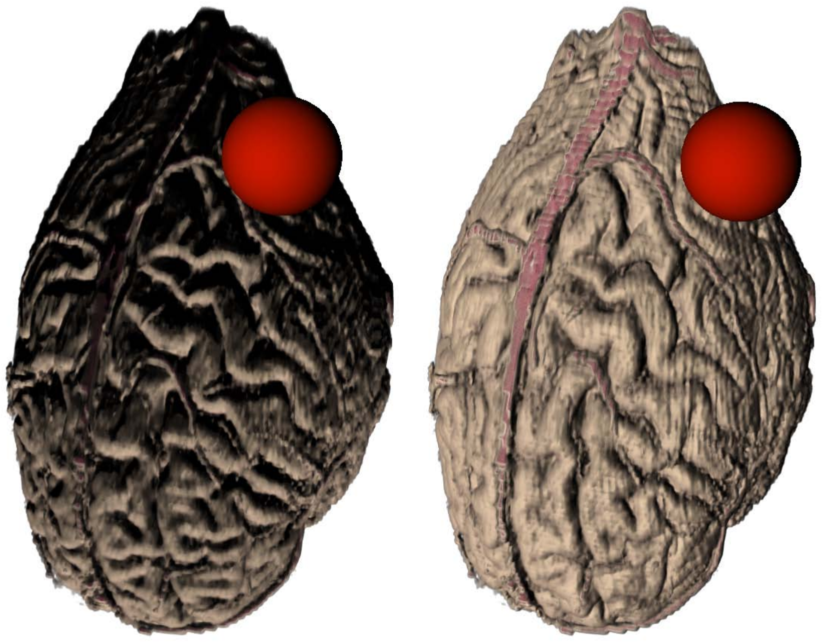


Figure 4.14: A deformed brain rendered without illumination correction (left) and with illumination correction (right). The red sphere is placed as a reference of the direction of the light and shows the correct lighting.

To correct the illumination, the gradient from the volume must be transformed to convert it from volume coordinates to the world coordinates. The deformation gradient ($\mathbf{F}$), described in Section 5.3.2, converts a point from volume coordinates (undeformed state) to spatial coordinates (deformed state), so by multiplying $\mathbf{F}^{-1^T}$ to the original gradient we can correct it to match the deformed state. The main drawback of this method is that $\mathbf{F}$ is constant within each tetrahedron resulting in a discontinuous gradient correction, as it can be observed in Figure 4.15.

In a similar way to vertex normals that can be interpolated inside the faces of polygonal models to achieve a smoother illumination, $\mathbf{F}$ can be calculated for each vertex of the tetrahedron and interpolated inside it. This creates a smooth transition of the illumination from one tetrahedra to another. Figure 4.15 shows an illustration of the effect. As many methods exist to calculate the per-vertex-$\mathbf{F}$s, in this study it has been chosen to

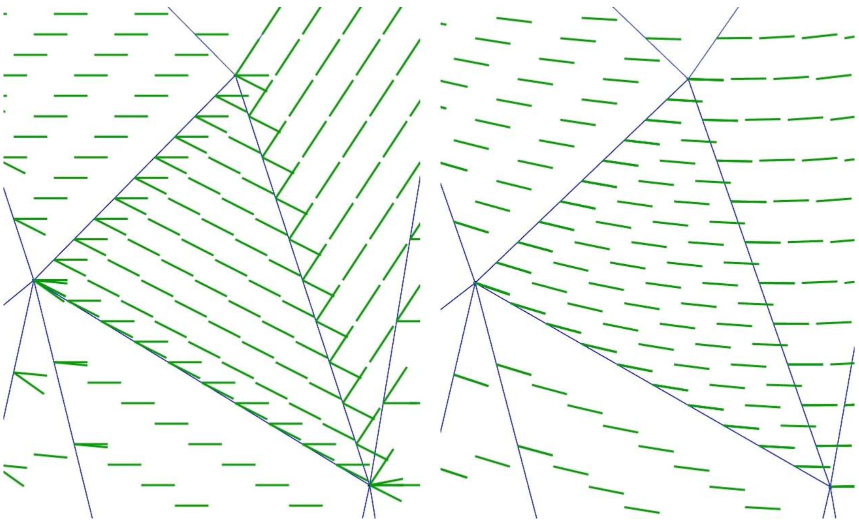


Figure 4.15: Simulation of gradient correction on triangles. Using constant **F** inside (left) to correct the normals creates abrupt changes in their directions. However, using interpolated **F** (right) creates a smooth transition of the direction of the normals between the triangles.

compute it outside the rendering pipeline to avoid interferences on the performance due to the method. The visual effects of the discontinuity and the smoothing can be observed in Figure 4.16.

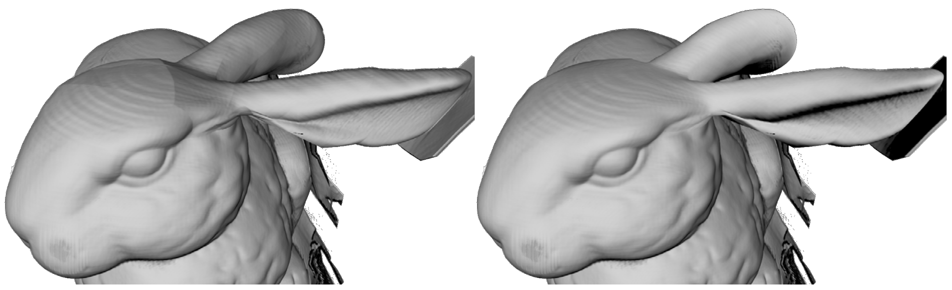


Figure 4.16: Comparison of a detail of the Stanford Bunny without (left) and with (right) smooth gradient correction. The difference can be seen in the ears of the bunny, which have been pushed downwards.

In terms of computational cost, when using a constant **F** it must be either calculated or read for each tetrahedron. In this last case, and depending on the implementation, it may imply three or four texture fetches.

When using interpolated gradients, the computational cost may vary considerably depending on whether the ray is smoothed or not. In the case rays are linear, **F** can be computed at the entry and exit points ($\mathbf{F}_{entry}$ and $\mathbf{F}_{exit}$ respectively) by interpolating it from tetrahedron's vertices, and then linearly interpolated along the ray:

$$\mathbf{F}(t) = (1-t)\mathbf{F}_{entry} + t\mathbf{F}_{exit} \tag{4.18}$$

On the other hand, if rays are smoothed (as described in Section 4.4.2) then the only solution is to perform a barycentric interpolation per contributing sample as described in Equation 4.19, where q are the barycentric coordinates of the sampling point.

$$\mathbf{F}(q) = \mathbf{F_0}q_0 + \mathbf{F_1}q_1 + \mathbf{F_2}q_2 + \mathbf{F_3}q_3 \tag{4.19}$$

Figure 4.17 shows the performance difference when using smooth rays. As expected, the impact of smoothing the **F** is very small when using direct volumetric isosurface rendering, as the calculation of the light is performed just once. As a result, if properly optimized, the interpolation is performed just once per pixel. However, it can be seen that the cost of smoothing the gradients when using FVR is considerable.

In order to reduce the impact of smoothing the gradients, another method can be used. Instead of deforming the gradients, the light and camera directions can be transformed using $\mathbf{F}^{-1}$. This correction not only creates an equivalent illumination but also allows the use of additional optimizations:

- In the case of a constant **F** per tetrahedra, the corrected light and camera directions are calculated once per tetrahedron instead of performing a matrix vector multiplication per contributing sample.
- In the case of using interpolated **F**s and linear ray segments, light and camera directions are calculated at the entry and exit points and interpolated along the ray, reducing the calculations to a linear interpolation of two vectors.

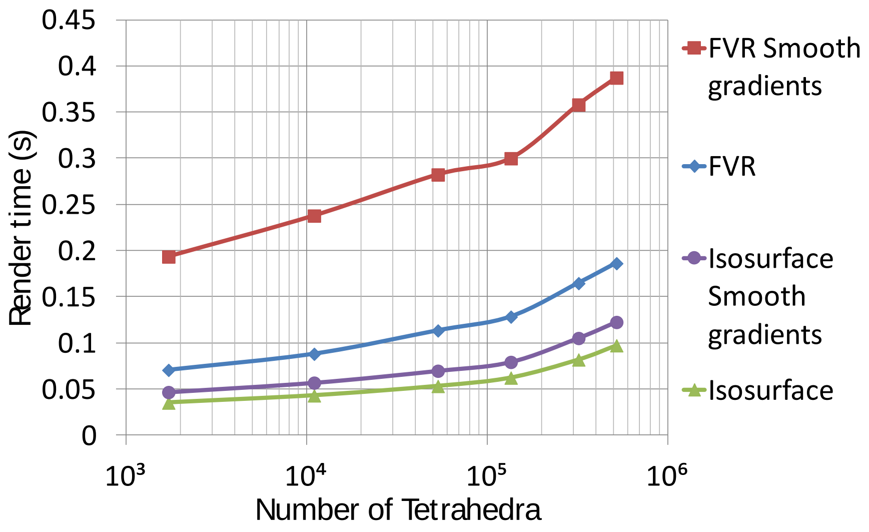


Figure 4.17: Chart showing the increase on rendering time when using smooth rays, with and without using smooth gradients in direct volumetric isosurfaces and FVR.

- In the case of interpolating **F** and using smoothed ray segments, instead of using four matrices to perform the barycentric interpolation (as shown in Equation 4.19), the corrected light and camera directions are used, reducing the necessary computation considerably.

The results of these optimizations can be clearly seen in Figure 4.18. Note that the improvement applies to all the methods described up to this point. The isosurface rendering has been let out of this chart as it has been shown in previous results that the effect of illumination cost are minimal in this render mode. Nevertheless the improvements shown here are also applied to it in a smaller degree.

Additionally, as this optimization deals with the illumination calculation, the number of samples performed for each ray is very significant. In Figure 4.19 the increase of the performance in the aforementioned three cases is shown with varying sample numbers and a constant tetrahedral model (with 12k vertices and 53k tetrahedra). As it can be seen, with a low number of samples the basic approach (constant **F** and linear ray) benefits most from the optimizations. However, as the number of samples increases the approach with smooth gradients and ray achieves

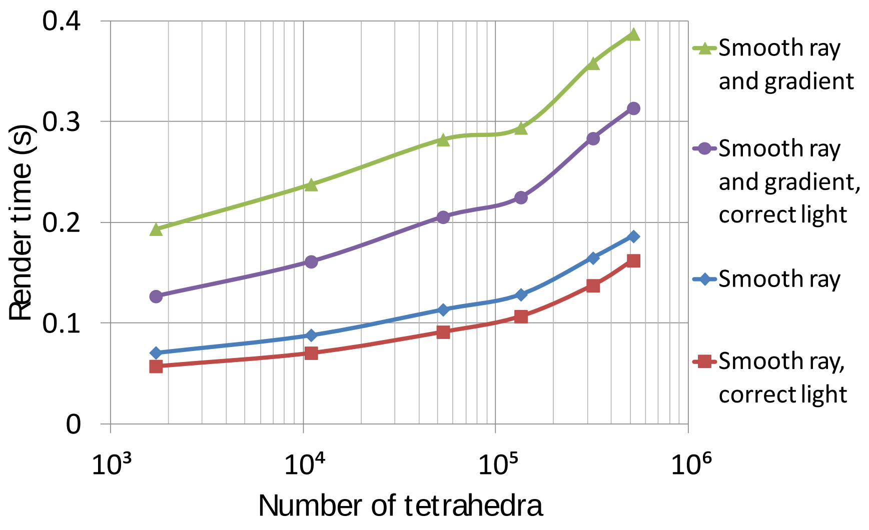


Figure 4.18: Comparison of FVR rendering times of different methods correcting gradients or correcting light and camera direction.

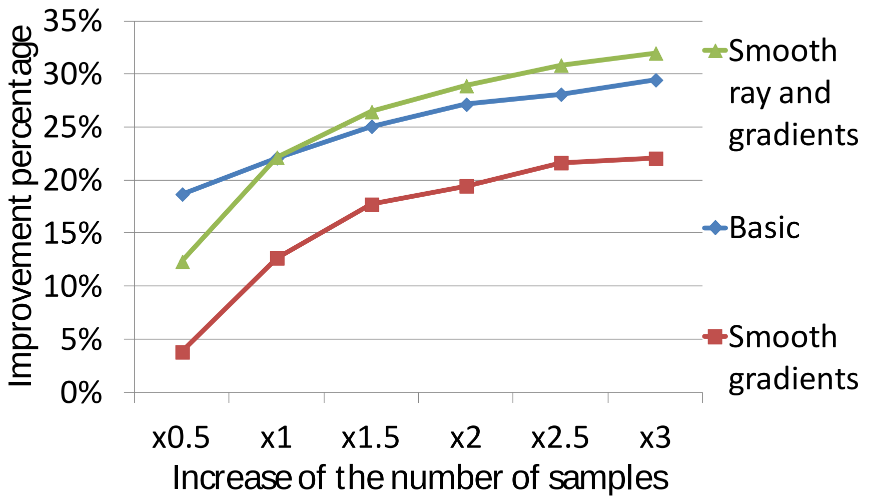


Figure 4.19: The different rendering methods' increase of performance by correcting light and camera direction with varying sample numbers.

a higher benefit. Nevertheless, this chart confirms the increase on the performance with the transformation of light and camera directions instead of the correction of the gradient. As the tests show, the improvement can be as high as a 32% increase of performance over non-optimized versions.

It can be seen that the correction of illumination is needed in order to have a realistic rendering of the object. The use of a constant **F** can be justified if a non uniform structured volume is used, as its gradient will vary enough. With this variation, if small or smooth deformations are performed, the use of a constant **F** can barely be noticed. Furthermore the cost of using the interpolated **F**s is large when using FVR.

On the other hand, if a uniform volume is used or large deformations take place, the artifacts created due to the constant **F** are noticeable. Using FVR the viability of smooth gradients will depend on the volume and user transfer functions. However, when using direct volumetric isosurfaces, the improvement is much more noticeable as shown in Figure 4.20, and the impact on the performance is minimal.

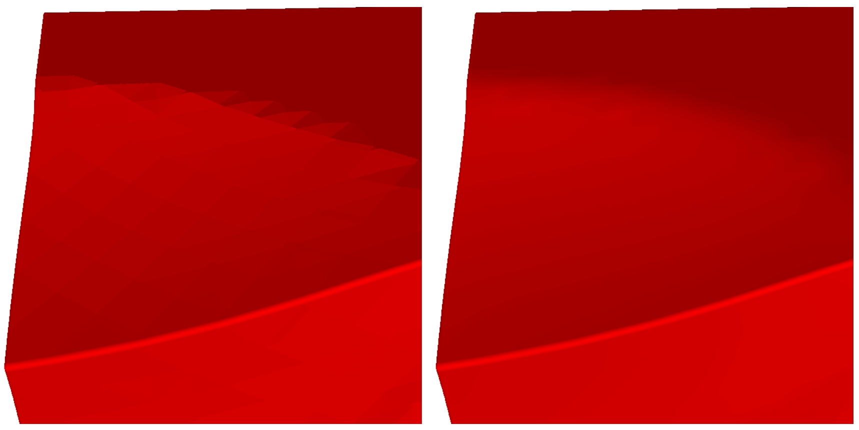


Figure 4.20: Comparison of an isosurface rendering of a deformed cube without (left) and with (right) smooth **F**s.

4.4.4 Further improvements

Many other improvements are possible on the ray traversal of deformable volumes, but they fail out of the scope of this dissertation. For example, thanks to the use of a traditional ray casting within each tetrahedron , most existing improvements for raycasting can also be used, such as adaptative sampling (Roettger et al., 2003) (as the means to achieve the desired steps have been presented) and empty space skipping (Kruger and Westermann, 2003). As the position of the samples in both volume space and world space is also known in the ray traversal, shadowing methods such as shadow mapping or deep shadow maps can also be directly used with the method.

However, some specific improvements or modifications of the raycasting pipeline are used in particular fields to achieve their objectives. For example, in the case of medical visualization applications, it is common to render only parts of a dataset for its exploration to improve the visualization. In these cases where this is performed using the usual raycasting method, such as in (Monclús et al., 2009), the presented method can be used without major modifications.

On the other hand, some methods cannot be used directly. This happens when the whole volume needs to be preprocessed and stored for its later use, as in volumetric ambient occlusion (a desirable feature for certain applications). In this case, as the on-the-fly processing of the necessary data is a very time consuming task, other options must be used. A good solution is the use of image space ambient occlusion algorithms, such as the one presented in (Bavoil et al., 2008) or more recently in (Díaz et al., 2010).

4.5 Rendering performance study

As shown in Section 4.3, there are many options and features that can be used in the visualization of deformable models. Although some of the choices have clearly surpassed the others, the choice of the rest of the options is not a trivial task. Additionally, the interactions between the methods also have to be taken into account when making this decisions. Therefore, a performance study has been done to drive these decisions.

The tests in this study have been performed in an end user application, in order to provide accurate and useful information. For this purpose all the renderings in the study used a viewport sized to 1000×1000 pixels, where the volumes occupied around 70% of the window space. In order to test the impact of the rendering with the tetrahedra, the volumes used in the test have all the same size, 200^3, which is around the volume sizes used in said application. The complexity of each model is shown in Table 4.1.

Vertices	0.5k	2.7k	12k	30k	69k	110k
Tetrahedra	1.7k	11k	53k	135k	320k	520k

Table 4.1: Table showing the details of the used models.

The testing machine is a Intel Core i7 CPU at 2.80 GHz with 3GB of RAM. The GPU is a GeForce GTX 470 with 1280MB of dedicated memory. The system runs MS Windows 7 (32 bit version).

As aforementioned, some methods have been proved to give better results so in this section only the best will be used, i.e., when talking about ray traversal ***ceil rounding*** will be used and the illumination will always be calculated transforming the light and camera direction.

Leaving those obvious choices aside, there are still two main options: the desired level of continuity of the ray and the use of constant gradients or smoothed gradients.Thus, the first option is using C^0 continuity and not smoothed illumination calculation, which we have called **Basic**. The second option is using G^1 continuity and not smoothed gradient correction, which in the charts is called **Smooth ray**. The third option is the use of C^0 and using smoothed gradient correction using the transformation of the light and camera direction, which we have called **Smooth gradients**. The last option is the combination of both methods, that is, using G^1 continuity and smooth gradient correction, called **Smooth ray and gradients**.

Additionally, the rendering mode to be used must be taken into account, as it has been shown that some options have a bigger impact (visually or computationally) in some cases. So both full volume rendering (FVR) and direct volumetric isosurfaces have to be tested with those four options.

Figure 4.21 shows the frames per second (FPS) achieved with the different methods using FVR. It can be seen that it is always a computationally demanding tasks, but by adequately choosing the method and quality, interactive frame rates can be achieved in most cases.

The chart shows that when using FVR rendering, an equilibrium between visual quality, speed and rendering framerate must be reached. This equilibrium will greatly depend on its intended use, but from the previous experiments some suggestions can be made. As aforementioned, the use of a higher level of continuity in the rays does not give a significant

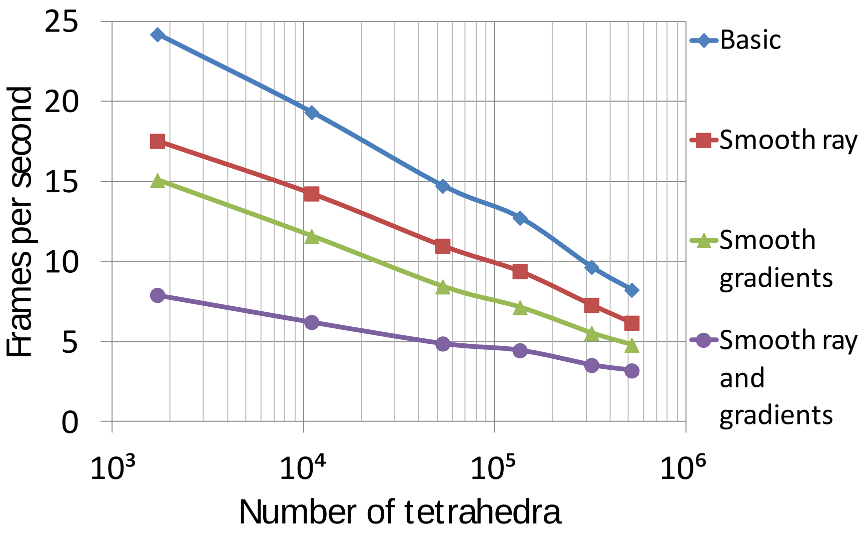


Figure 4.21: Chart showing the frame rate obtained with the different options with FVR.

quality increase in this case. Having that into account, as well as the cost increase when using smooth rays alone or combined with smooth gradients seen in the chart, the use of smoothed rays with FVR should be the first option to disable if needed.

In the case of smooth gradients, they have demonstrated to deliver an important visual quality increase. This increase is tightly tied to the size of the deformations, being less noticeable with small deformations. However, the chart shows that smooth gradients are an important burden in the rendering pipeline. In this case, a choice between quality and rendering speed must be made, taking into account the size of the tetrahedral model as well as the size of the viewport (it should be noted that the viewport used for these tests is bigger than common volume rendering windows). Additionally, as the visual improvement is tied to the size of deformations, the expected deformations should be taken into account when making this decision.

On the other hand, direct volumetric isosurface rendering is able to achieve interactive rates with nearly all tested models, as shown in Figure 4.22. This fact, coupled with the aforementioned higher quality, makes direct isosurface rendering a good alternative to extracted isosurfaces or polygonal rendering of the tetrahedra. The direct volumetric isosurface rendering performed for these tests include the iterative intersection refinement (set to four iterations).

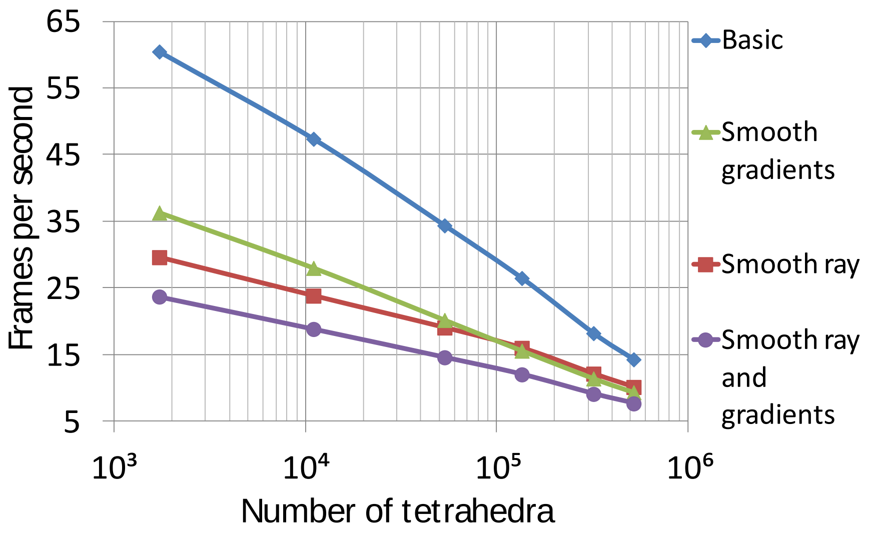


Figure 4.22: Chart showing the FPS of the different options with isosurface rendering.

It can be seen that direct volumetric isosurfaces enable the use of the options more easily. In a similar manner to the options with FVR gradient smoothing is the one with highest visual impact but, in contrast to FVR, isosurface rendering calculates the lighting only once, greatly reducing the cost of smooth gradients.

Another difference respect FVR is the effect of smoothed rays, which have a noticeable impact when isosurface rendering is used. However, as ray smoothing has to be performed through the whole ray traversal, it does not benefit from direct volumetric isosurface rendering as much as FVR. Taking these facts into account, the use of both smooth gradients and smooth rays is recommended when possible. And, if the conditions of the application or the model do not allow it e.g. the application must reach high frame rates or the models are very big, smooth gradients should be preferred over smooth rays, as they have a higher visual impact and benefit from a reduced computational cost with models up to medium size compared to smooth rays.

For the sake of completeness these tests where also performed with varying sample numbers. Figure 4.23 shows the frames per second achieved with the model of 135k tetrahedra and a structured volume of 200^3 using FVR with different combinations of the presented methods. It provides an accurate example of the scalability of these algorithms in terms of the resolution of the structured volume.

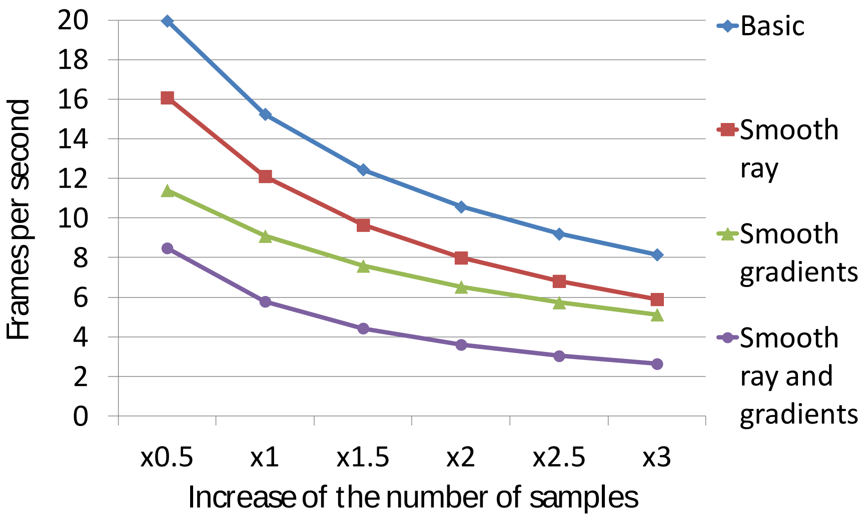


Figure 4.23: Chart showing the FPS with a varying sampling rate using FVR with different combinations of the presented methods. The used tetrahedral model has 135k tetrahedra and 30k vertices.

This chart shows that smooth gradient calculations are slightly less affected by an increase in the number of samples along the ray in comparison to smooth rays. This comes from the fact that smooth gradient calculations add more calculations per tetrahedron and not that much per sample, in contrast to smooth ray calculations, which are more dependant on the number of samples. When using direct volumetric isosurfaces, this difference becomes more acute as it adds to the natural advantage of smooth gradients in this method.

The tests described in this chapter have been performed over synthetic models, so some variation in these numbers may appear with real data. To give a better outlook on the possibilities, Figure 4.24 shows the visual result of some models rendered using the studied options as well as their frame rate.

The images have been rendered with a viewport of 700×700. All these images have been created correcting the light and camera direction as well as using stochastic jittering. The used models' details and rendering modes are shown in Table 4.2, in descending order.

Dataset name	Rendering mode	Number of vertices	Number of tetrahedra	Dimensions of structured volume
Synthetic 1	Isosurface	2.7k	10k	200^3
Stanford Bunny	FVR	4.1k	13k	$512 \times 512 \times 361$
Brain	FVR	7k	40k	$216 \times 128 \times 142$
Lung	FVR	7k	36k	$229 \times 275 \times 241$

Table 4.2: Table showing the details of the models used in Figure 4.24.

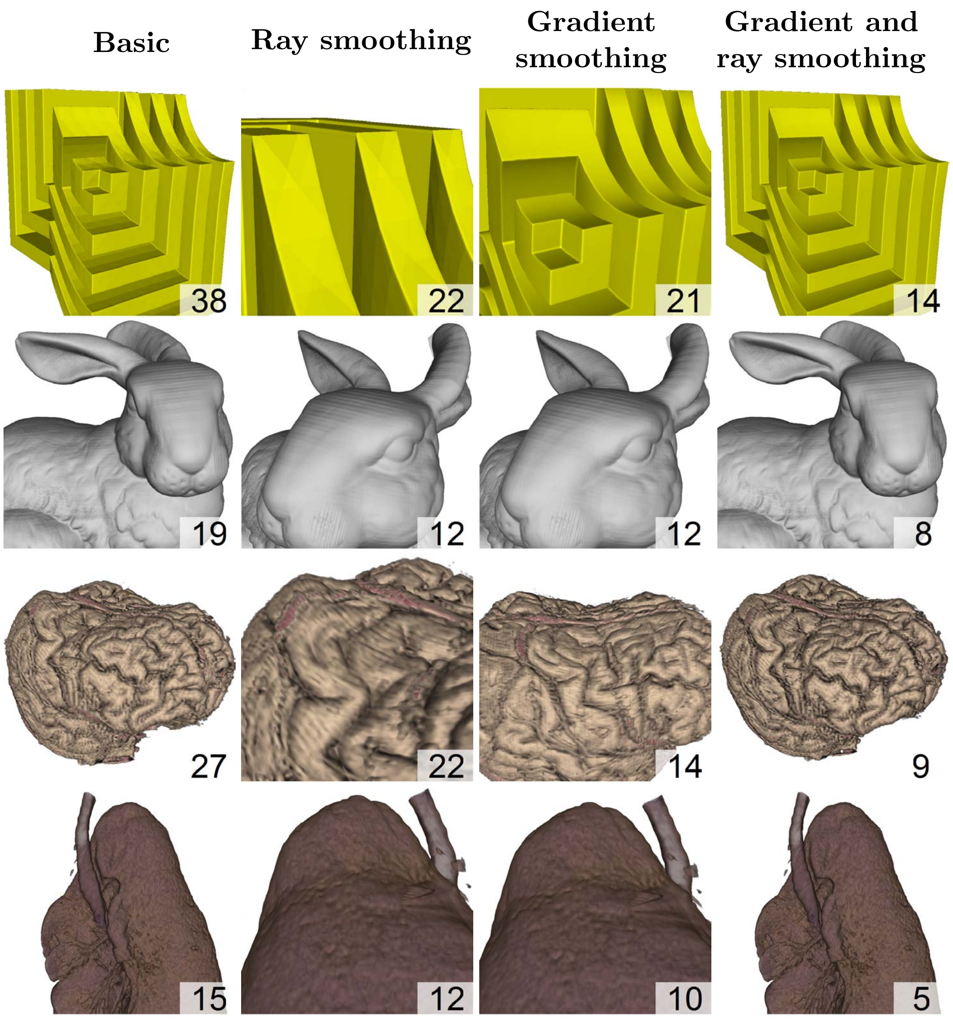


Figure 4.24: Demonstration of different models using the studied raycasting techniques. The frame rate shown in each image is the one achieved in that specific configuration and viewpoint. Details of the models can be found in Table 4.2.

4.6 Discussion

This chapter has presented and studied a method to visualize a deformable volume using volumetric rendering. The proposed method avoids resampling altogether, while maintaining a high image quality and good performance. And, thanks to the use of a tetrahedral mesh for the deformation, the method is specially suitable for applications with physical simulations such as surgery simulators, which usually have to resort to polygonal rendering. This chapter has presented and studied a method to visualize a deformable volume using volumetric rendering. The proposed method avoids resampling altogether, while maintaining a high image quality and good performance. And, thanks to the use of a tetrahedral mesh for the deformation, the method is specially suitable for applications with physical simulations such as surgery simulators, which usually have to resort to polygonal rendering.

Additionally, different improvements on quality and performance have been developed. Using these methods, the quality of the rendering can be improved by increasing the continuity of the ray or using a smooth illumination, akin to the use of Goraud shading instead of flat shading in polygonal rendering.

Furthermore, performance tests have been conducted for the different options available and their combinations. These tests give the necessary information to make good decisions about the available visualization options, in order to get the best quality while conforming with the restrictions of the objective application.

Summarizing, the presented method and its study allow the use of raycasting for applications with deformable models with a method that is easy to modify and adapt, giving the developers the freedom to choose the level of quality and the improvements to be used in it.

Chapter 5

Neurosurgery Simulator

The art of medicine was to be properly learned
only from its practice and its exercise.
Thomas Sydenham

Part of this chapter has been published in:

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Simulation and visualization techniques for a surgery simulator of skull drilling and brain interaction tasks". IEEE Computer Graphics and Applications, 2013. Accepted.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization". In Proceedings of the XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012). San Sebastián, Spain. November 19-21, 2012.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data". In Proceedings of the XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011), pp. 423–426. 2011. Cáceres, Spain. November 16-18, 2011.

5.1 Introduction

This dissertation is part of a bigger project whose main goal is the development of a realistic and immersive neurosurgery simulator oriented towards brain tumour resection. Although the development of the complete neurosurgery simulator falls outside the scope of this dissertation, three intermediate prototypes have been developed:

- Craniotomy simulator: This stage deals with the first part of the procedures, drilling the skull in order to access the brain. In this stage rigid models are used, so rigid volumetric visualization and rigid model haptic interaction are needed.

- Brain interaction simulator: This stage deals with the interaction with the brain, which is restricted to touching and pushing. In this stage the behaviour of the tissues will be simulated, so deformable volumetric visualization and deformable model haptic interaction are needed. Additionally, the deformation of the brain must be calculated in real time.

- Integrated simulator: In order to develop a combined simulator prototype, both simulation modes have to be integrated into a single pipeline. This simulator will be used as a first step in the development of a full neurosurgery simulator.

As it can be observed, the first two prototypes have very specific visualization needs. The methods necessary to fill said needs have been presented in the previous sections. This chapter will present how these methods have been used in each stage and why. Additionally, although the main focus of this dissertation is the visualization, some of the other modules for the simulators developed in the department will be briefly explained.

In terms of hardware, the presented prototypes use the same machine as in Chapter 4: A a Intel Core i7 CPU at 2.80 GHz with 3GB of RAM with a a GeForce GTX 470 GPU with 1280MB of dedicated memory running on MS Windows 7 (32 bit version).

5.2 Craniotomy simulator

Neurosurgery procedures are complex and delicate surgeries which need a precise control of the tools and a vast knowledge. Existing procedures are very varied and can be radically different. Craniotomy, however, is the first step of most neurosurgery procedures as it gives access to the brain by drilling the skull in the appropriate place. It can be seen that skull drilling is an important procedure, requiring a steady hand and a precise control of the drilling depth, in order to avoid unnecessary damages to the underlying tissue.

Medical simulators can be divided in six stages, image segmentation, 3D visualization, physical modelling, physical simulation, collision handling and haptic feedback. This section describes the steps given to perform these stages, including different disciplines such as visualization, collision detection, visual feedback and haptic feedback. A real time volumetric framework has been developed for its use in a craniotomy simulator, enabling the use of patient specific data.

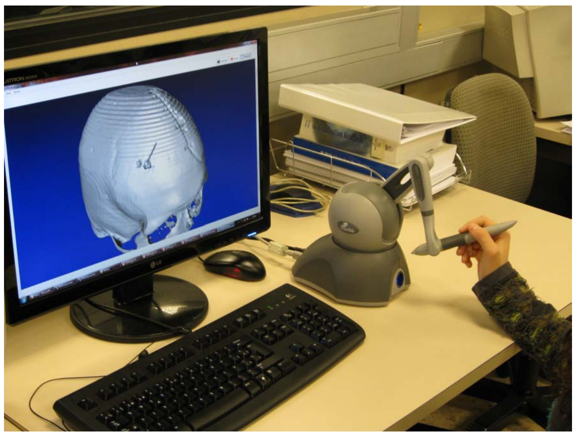


Figure 5.1: Craniotomy simulator configuration.

The developed craniotomy simulator simulates a craniotomy procedure using a haptic device to control a virtual drilling tool. The system enhances the immersion when drilling bone through high quality visualization and haptic response. It provides visual and haptic feedback correspondent to an actual craniotomy intervention. Figure 5.1 shows the configuration of the simulator.

The Craniotomy Simulator involves the design and development of the following areas: interface, haptics, collisions and visualization. In order to cope with all the necessary calculations these needs have been separated in two threads. The main thread generates all the components, including the second thread, and manages the interface and visualization. The second thread, the haptic thread, runs separately in the background managing the haptic device and the connection with the collision module. Figure 5.2 shows a schematic diagram of the system architecture.

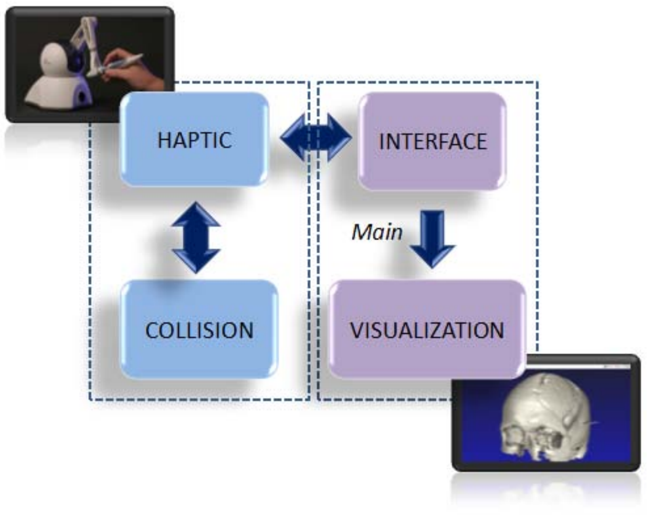


Figure 5.2: Diagram of the craniotomy simulator.

Although this thesis is focused in the visualization, a brief explanation of the other parts of the simulator will be given in order to allow a broader view of the issues and their solutions.

5.2.1 Haptic feedback and collision detection

The objective of the haptic module is to manage the user input from the haptic device, and respond adequately to the interaction. For this purpose, it must communicate the input to the collision detection module and calculate the matching force feedback from the its result. The haptic device used in the craniotomy simulator is a Phantom Omni with 6 DoF[1] input

[1]Degrees of Freedom

and 3 DoF force feedback. As this haptic only respond with translation forces, the calculation of the rotating forces can be avoided.

This module reads the input from the haptic device, and resolves it to a position in space. This position is then fed to the collision detection algorithm, which returns the position the virtual tool should adopt as well as the normal of the collision, if any.

With the correct position calculated, the virtual tool is updated to reflect said position visually. In addition, the magnitude and direction of the force to be returned to the user is calculated and fed to the haptic device. All these calculations must be performed in less than 1 millisecond so that the forces are returned with a frequency higher than 1000 Hz in order to give a good haptic response.

5.2.1.1 Collision detection

The collision module is responsible of detecting any intersection between the virtual tool and the skull. Additionally, it must calculate the direction of said collision so that a correct force feedback can be calculated.

The research and development of the algorithm used in the craniotomy simulator is one of the focuses of the dissertation by (Echegaray, 2012), and further information about these topics are available in it. It should be noted that, in contrast to the majority of other simulators, this collision detection algorithm does not convert the data to polygonal surfaces. Instead, it works directly with the volume, retaining the complete dataset.

5.2.2 Volumetric visualization

In previous chapters different volumetric visualization algorithms have been presented. In the craniotomy simulator, only rigid models are used, as only rigid interactions are considered for the drilling. As a result, the algorithms presented in Chapter 3 are used in order to achieve the best quality possible.

The patient data used to visualize the skull is a CT, as it has a very good contrast for the bones. Furthermore, the renderization of the skull is opaque, so further optimizations can be used.

Taking all the aforementioned facts into account, it is easily seen that the use of direct volumetric isosurfaces for the skull is a very good choice as it allows the use of high quality visualizations at very fast speeds. Additionally, thanks to the fact that in terms of depth calculation direct volumetric isosurfaces behave as regular polygons, another volume can be added to the visualization without any changes in the pipeline.

Thanks to this, the simulator can visualize a skull from a CT with direct volumetric isosurfaces and a brain from a MRI inside it with full volume rendering (FVR). This achieves a great quality visualization with a reduced computational cost.

The casting of shadows into the volumes by polygonal models further increases the visual quality and, at the same time, greatly enhances the depth perception and control of the user.

The resulting visualization can be observed in Figure 5.3, where different parts of the craniotomy procedure are shown.

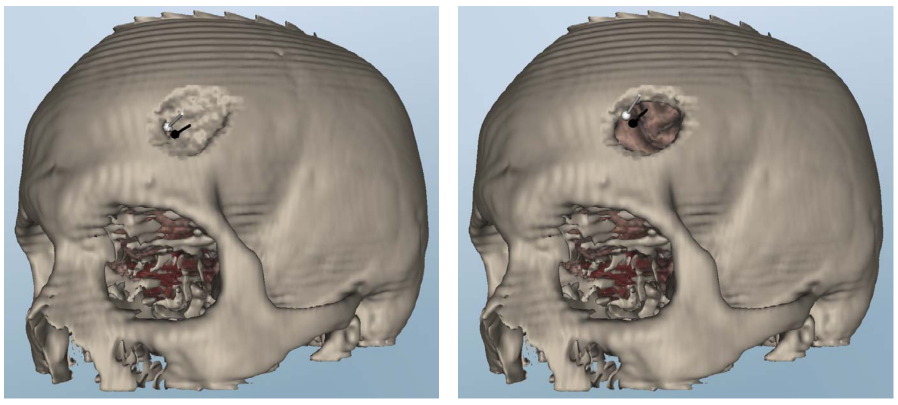

(a) (b)

Figure 5.3: Two captures of the skull drilling process: (a) starting the drilling and (b) the brain exposed.

5.2.3 Graphical user interface

The graphical user interface (GUI) of the simulator at first presents a four panel view, showing the CT of the patient from three points of view (sagittal, axial and coronal) and the volumetric visualization in one screen (Figure 5.4).

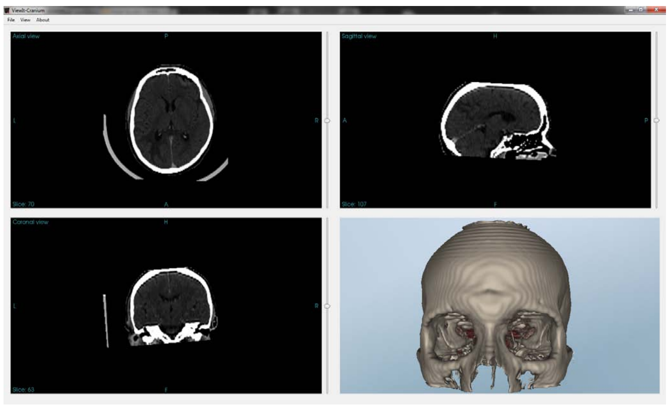


Figure 5.4: Main screen of the craniotomy simulator.

After the user has planned the entrance, the simulator will change to a full screen mode where only the volumetric visualization is visible. Once there the drilling process starts and the user chooses the desired drill size through an interface operated with the haptic device, seen in Figure 5.5. It should be noted that the drill size can be change in any moment.

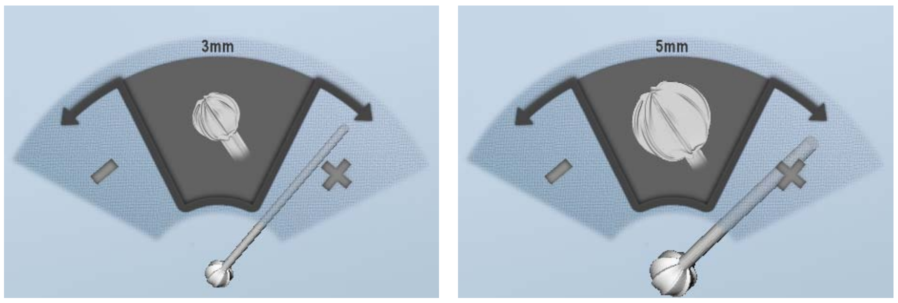


Figure 5.5: Drill changing interface, operated by rotating the haptic device.

5.3 Brain interaction simulator

After gaining access to the brain through a craniotomy, the surgeon interacts with the brain. In this prototype the interaction of a surgeon with a patient specific brain is simulated, allowing the user to touch the brain trough the use of a haptic device.

In order to realistically and accurately interact with the brain, a physical simulation is needed to calculate the resulting deformations. Additionally, this prototype must deal with a physical deformation of the patients brain so specific algorithms are needed to deal with the changing volume. It can be seen that the necessary changes greatly increase the computational cost, so the architecture must be redesigned to fit these needs.

In this prototype, each main module has its own thread in order to take advantage of the actual multi-core central processing units as it can be seen in Figure 5.6. It should be noted that this simulator is very GPU intensive, as both the simulation and the rendering are calculated with it.

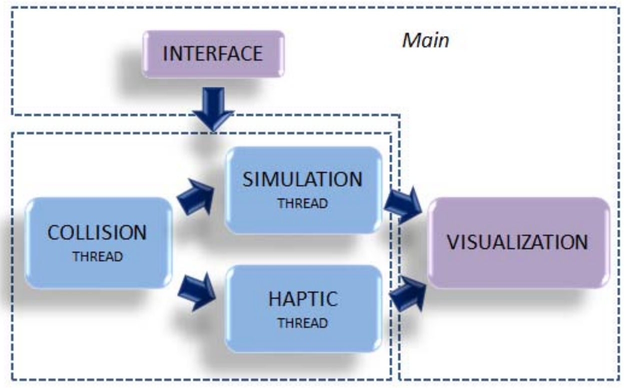


Figure 5.6: Diagram of the brain interaction simulator.

5.3.1 Haptic feedback and collision detection

The haptic module serves the exact same purpose it does in the craniotomy simulator. Its job is to manage the haptic devices input and feed it to the visual and collision detection modules.

In contrast to the haptic thread presented in the craniotomy simulator, in the brain interaction simulator each module runs in its own thread as a result of the increase on the computation time needed for the collision detection.

This increase comes from the change on the collision detection primitive (which are tetrahedra in this case) and the added needs due to a deformable collision. More information and details about the collision detection module can be found in (Echegaray, 2012).

5.3.2 Physical simulation

In order to reach the necessary realism, the interaction of the user must trigger a physical response on the brain. Additionally, as the objective is an accurate medical simulation, this response must be as physically realistic as possible.

The most relevant algorithm, when accuracy in the simulation is an important factor, is the Finite Element Method (FEM) (Bathe, 1996). Traditionally, computer graphics applications used implementations based on linear elasticity. These approach is suitable for the simulation of objects that undergo small deformations and whose material behaviour can be considered linear. The behaviour of soft-tissue, such as brain, is notoriously non-linear, and in many cases large deformations are also common. Because of these reasons, a non-linear FEM simulation with a tetrahedral mesh is preferable in this simulator.

The non-linearity is handled using a total Lagrangian non-linear explicit dynamics (TLED) approach (Irving et al., 2004). In a Lagrangian approach, the dynamic analysis is performed by tracking the material particles forming a body. In particular, two sets of coordinates can be defined, the spatial coordinates $\mathbf{x}$ which represents the position of one particle in the deformed state, and its original position or material coordinate $\mathbf{X}$. Either representation can be used to identify each material particle. In a total Lagrangian representation, all forces, deformations and material stresses are expressed in the material coordinate system $\mathbf{X}$.

The deformation gradient tensor is a basic measure of the local deformation of the material $\mathbf{F}_{ij} = \frac{\partial x_i}{\partial X_j}$. Once this value is computed other deformation measures can be computed directly. For example, the Green-Lagrange strain tensor can be written as $\mathbf{E} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} + \mathbf{I}\right)$. In this case, the interpolation of the deformation is linear within the tetrahedron and stress is constant. In this case, it can be demonstrated (Irving et al., 2004) that $\mathbf{F}$ can be expressed as $\mathbf{F} = [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1]\,[\mathbf{X}_2 - \mathbf{X}_1, \mathbf{X}_3 - \mathbf{X}_1, \mathbf{X}_4 - \mathbf{X}_1]^{-1}$ where $\mathbf{x}_i$ are the deformed positions of the nodes of the element and $\mathbf{X}_i$ are their material coordinates (i.e. their initial position). The second part of this expression can be precomputed and stored in memory to improve the performance of the method.

In TLED elastic forces have to be computed with regards to the initial configuration of the body. This leads to an expression of the elastic forces affecting a node of an element a such as (Irving et al., 2004):

$$\mathbf{f}_a^e = \int_{\Omega_m} \mathbf{P}\frac{\partial {N_a}^T}{\partial \mathbf{X}} d\mathbf{X} = \int_{\Omega_\xi} \mathbf{P}\frac{\partial {N_a}^T}{\partial \mathbf{X}} d\left|\frac{\partial \mathbf{X}}{\partial \xi}\right| \xi \tag{5.1}$$

where, Ω_m and Ω_ξ represent the volume of the element in either the global framework and in an ideal system based on isoparametric coordinates. N_a represents the interpolation function for node a, $\mathbf{X}$ the material coordinates of a particle, and $\mathbf{P}$ is the first Piola-Kirchoff stress tensor, which relates the deformation of the element and the mechanical stress in the material, expressed in material coordinates. Except for $\mathbf{P}$, all the values of the integral are referred to the initial configuration of the element and, again, can be precomputed and stored in memory for future use.

The value of $\mathbf{P}$ can be obtained from the deformation gradient $\mathbf{F}$ using the constitutive law of the material. $\mathbf{P}$ is constant within tetrahedron and the integrals can be easily computed, leading to a force generated by each element in each node a equivalent to (Teran et al., 2003):

$$\mathbf{f}_a^e = -\frac{1}{3}\mathbf{P}\sum_{i \neq a} A_i \mathbf{N}_i \tag{5.2}$$

where $A_i\mathbf{N}_i$ are the area weighted normals of the faces of the tetrahedron incident in node a in the original position which can be precomputed.

Adding the contributions from all the elements, the resulting elastic forces depend directly on the nodal displacements $\mathbf{f}_a = \mathbf{K}_a(\mathbf{u})$ through the first Piola-Kirchhoff strain tensor, and therefore, through the constitutive law of the material. In this work, the St Venant-Kirchhoff material model is used for simplicity, but more complex material models can be easily handled.

Due to the non-linearity of the motion equations, its time integration is specially well suited for explicit integration schemes, where the computation is performed with quantities defined in the current time step and with a known configuration of displacements. In this work a semi-implicit Euler integration is used. This scheme is conditionally stable, therefore, the integration time step has been set to a value where stability is guaranteed.

Thanks to this approach the algorithm can be easily parallelized, since all computation can be performed independently for each element. This enables an easy implementation on parallel systems and in particular in GPUs, as in this case, in which the physical simulation is performed in the GPU using CUDA.

Figure 5.7 shows a flow chart explaining the resulting algorithm. It can be seen that two very distinct phases exist, the calculation of the forces on the nodes performed for each tetrahedra and the calculation of the resulting force and consequent movement per each node.

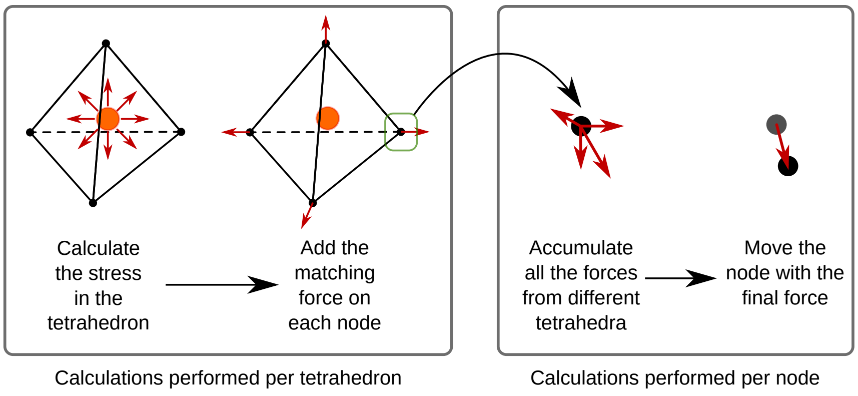

Calculate the stress in the tetrahedron
Add the matching force on each node
Accumulate all the forces from different tetrahedra
Move the node with the final force
Calculations performed per tetrahedron
Calculations performed per node

Figure 5.7: Diagram of the FEM simulation algorithm.

It can be seen that although each tetrahedron is completely independent in the calculation of the stress the accumulation of the forces on the nodes is not. As many tetrahedra will try to accumulate their forces in one node, a race condition arises, where a tetrahedron can overwrite the value of a node in the middle of an operation rendering it incorrect. To solve this race condition, three different options exist.

The first, and most straightforward, method is the use of atomic operations (operations that cannot be interrupted, introduced in CUDA 1.1) avoiding the problem altogether. Although this method performs better than barriers and locks, it still creates delay in the operation.

As the use of atomic operations slows down the simulation altogether, another alternative must be used. The second method overcomes this hurdle by avoiding the accumulation in the calculation of the tetrahedrons. To do this, this method stores the contribution on each node per tetrahedron by creating an array per node with one element per tetrahedron in which the node takes part (Figure 5.8).

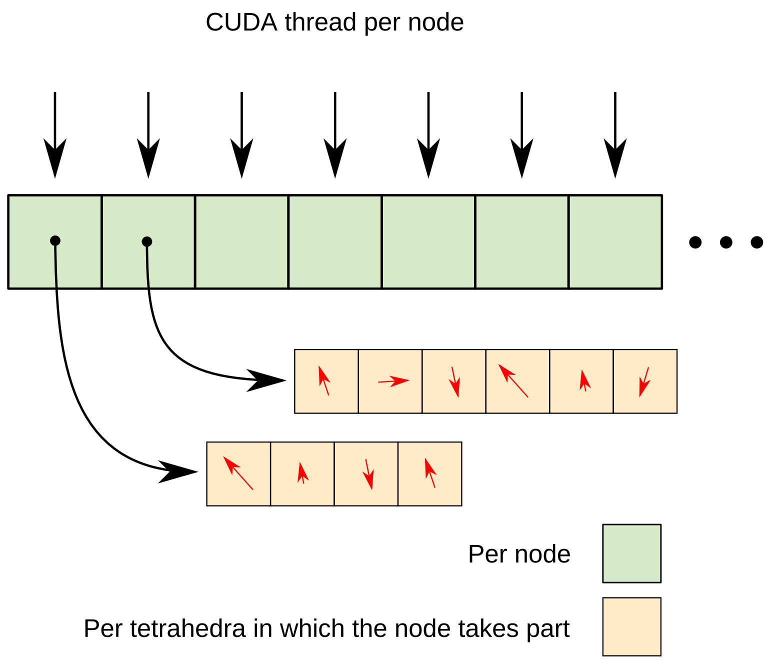


Figure 5.8: Diagram of the first possible structure to avoid racing conditions.

This removes the problem from the tetrahedral phase, and moves it to the node phase where no racing condition exists, but another problem arises. CUDA uses one thread per element, nodes in this phase, and by prefetching the adjacent memory positions it can greatly reduce the needed time. However, as the used memory structure does not store values adjacently, as seen in Figure 5.8, the calculation is greatly slowed down.

In order to take full advantage of CUDA, the structure must be reorganized so that the values fetched by the nodes are adjacent (called coalesced memory). This can be done by storing the values in a matrix where the values can be stored adjacent in memory, as seen in Figure 5.9.

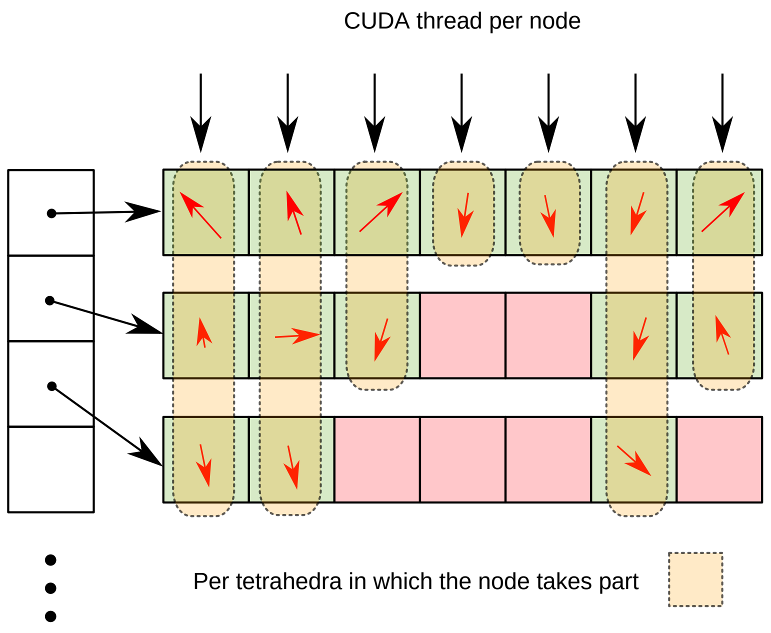


Figure 5.9: Diagram of the second possible structure to avoid racing conditions. The green squares are allocated memory storing information, and the red squares are allocated memory with no information.

Using this structure, the race condition is avoided and the memory is mainly coalesced, allowing very fast calculation times. On the other hand, part of the memory allocated does not store any information, so it allocates more memory than the strictly needed. However, with the used tetrahedral models this has not proven to be a problem, as the modern GPUs have high quantities of memory. Thanks to this methods speeds of less than 0.2ms with models with 12k tetrahedra (without volumetric rendering) can be achieved.

5.3.3 Volumetric visualization

In the brain interaction simulator, only deformable models are used, as deformable interactions are the focus of the simulator. As a result, the algorithms presented in Chapter 4 are used in order to achieve the best quality possible.

In this simulator, the patient data used as the input is a MRI as it provides a great contrast for the different tissues in the brain. This structured volume is then manually segmented to extract the brain. Additionally, once the brain is segmented a tetrahedral mesh is constructed to match it with a custom application implemented using *iso2mesh*[2].

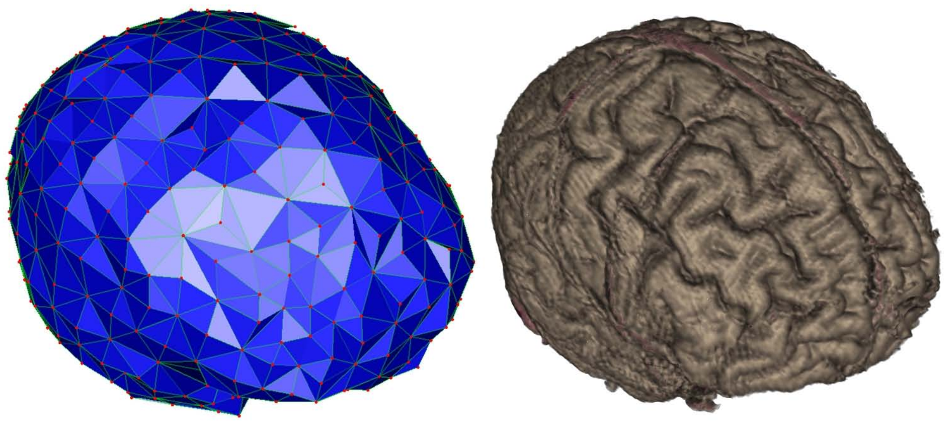


Figure 5.10: The tetrahedral model of a brain and the resulting visualization.

With the segmented brain as the structured volume and its matching tetrahedral mesh as the unstructured mesh, the previously presented methods can be used in order to achieve a high quality rendering in real time (Figure 5.10).Additionally, this tetrahedral mesh is also used for the physical simulation (Explained in previous Section, 5.3.2). As it can be seen, a FVR rendering mode is used, as the brain cannot be adequately rendering with isosurfaces.

Furthermore, as the simulation is performed in the GPU, there is no need to upload the updated model to the GPU in order to render it. By

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

just copying it from CUDA to the necessary graphic storage the rendering algorithm can have the correct data with no or little overhead. However, it should be noted that even though the continuous upload of the nodes can be avoided, a periodic download from the GPU cannot be completely avoided as the collision detection algorithm runs in the CPU.

As a result of all the calculations performed in the GPU, the quantity of processing time used by the rendering process must be as low as possible. Additionally, the usual interaction with the brain will only deal small deformations to the brain.

Taking all these into account, smooth gradients are not used in the simulator, as its use greatly increases the rendering times. Additionally, as the gradients of the brain are not uniform, by using a FVR rendering mode the artifacts created by the use of constant **F** are smoothed.

Moreover, as FVR is used and only small deformations are expected, the use of smooth rays will not improve the rendering quality in a noticeable manner. It can be deduced then that their use will increase the rendering times without major improvements. As a result, their use is not recommendable in this case.

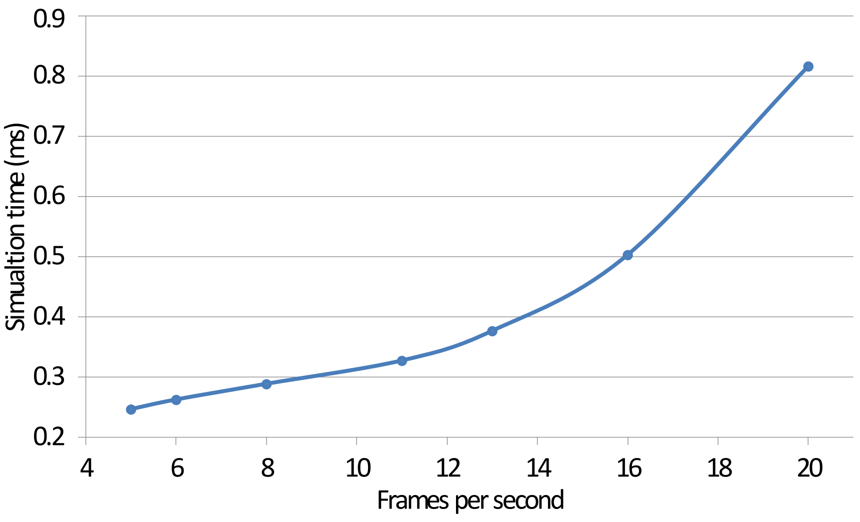


Figure 5.11: A graph showing the needed simulation calculation time per step with increasing FPS in a 1680 × 1050 viewport.

Using this algorithm frame rates higher than 20 frames per second

(FPS) can be achieved with 1680 × 1050 resolution. But as it has been previously said, this simulator uses the GPU heavily, and the visualization and the physical simulation must share the same computational time. To illustrate this problem, Figure 5.11 shows a graph with the increase of the time needed to calculate a simulation step due to the increasing FPS of the volumetric rendering of the deformable model. It is easily seen that a compromise must be reached, for which the time step needed for the model in the physical simulation has to be taken into account.

5.4 Integrated simulator

The last stage of the prototypes consist in the integration the craniotomy simulator (Section 5.2) and the brain interaction simulator (Section 5.3). This integrated simulator is still included within the early stages of the development of a full fledged neurosurgery simulator.

The integrated simulator offers a seamless experience by allowing the user to first drill the skull and then interact with the brain inside. This experience is further enhanced by the use of a haptic device for the interaction, improving the presence. Additionally, the high quality interactive visualization of the patient specific models immerses the user, completing the experience. The GUI of the final simulator prototype is shown in Figure 5.12.

Both visualizations (the rigid and deformable volumetric visualizations) could be integrated without many modifications. As the skull is rendered using direct volumetric isosurfaces, it will occlude the rest of the objects using the same pipeline as the polygonal objects. This way, the addition of the second volume to be rendered is done without problems, as it is the only object requiring special depth checking. Figure 5.13 shows a diagram of the union of the two visualization modes.

Additionally, as the access to the brain is the drilled hole, the part of the brain rendered is comparatively small. This is specially important because the brain is rendered using FVR, and, as explained in Section Section 5.3, the GPU is also needed for the physical simulation.

The integration of the collision detection method, however, requires special treatment as the used primitive changes. For this purpose, both the drilling task and the interaction with the brain have a flag to inform whether

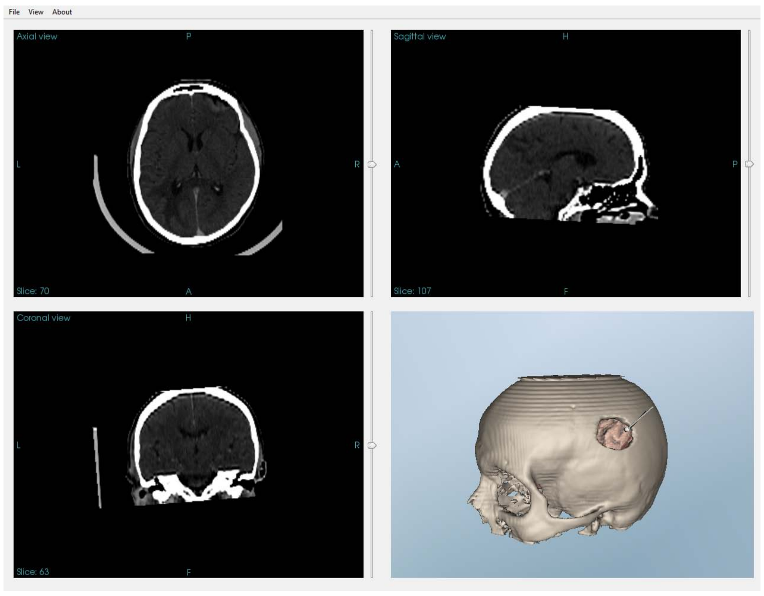


Figure 5.12: GUI of the integrated simulator.

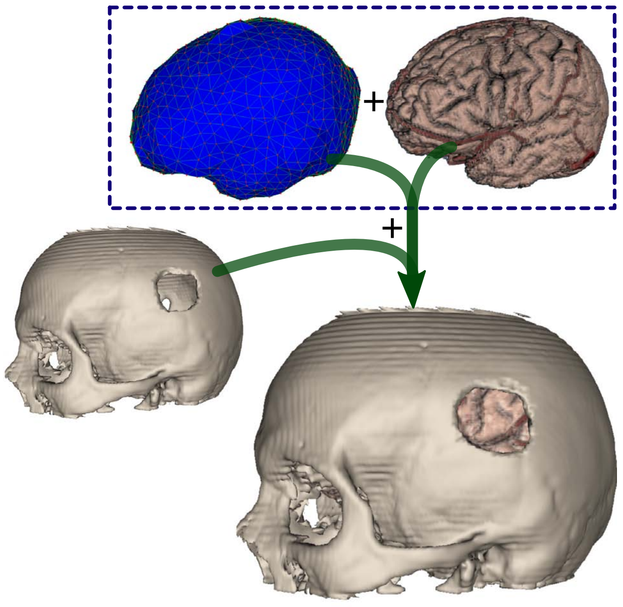


Figure 5.13: A diagram of the union of the rigid and deformable volumetric visualization.

one or the other is being performed. With this information, the algorithm activates one module or the other. It must be noted, however, that the deformable collision detection algorithm is active when the craniotomy flag is active. This way, the user can be warned when the drill is touching the brain, although it is not deformed at this stage as the simulation is not active. More detailed information about this integration and the additions can be found in (Echegaray, 2012).

The final integrated simulator, the result of the fusion if the craniotomy simulator and the brain interaction simulator, presents the user with patient specific skull and brain models. It allows the user to perform a virtual craniotomy to the patient (Figure 5.14) and interact with its brain (Figure 5.15), making possible its use as a training platforms for students or a as rehearsal platform for surgeons.

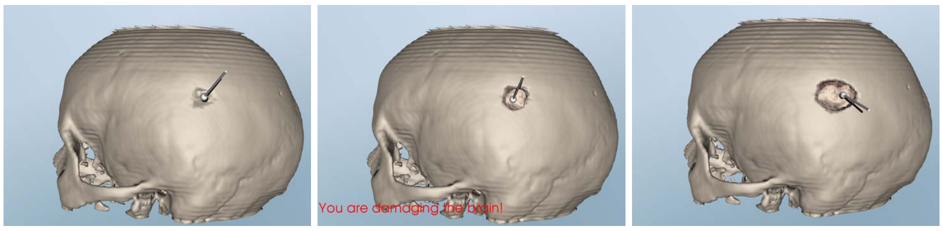


Figure 5.14: Sequence showing a drilling work in the integrated simulator.

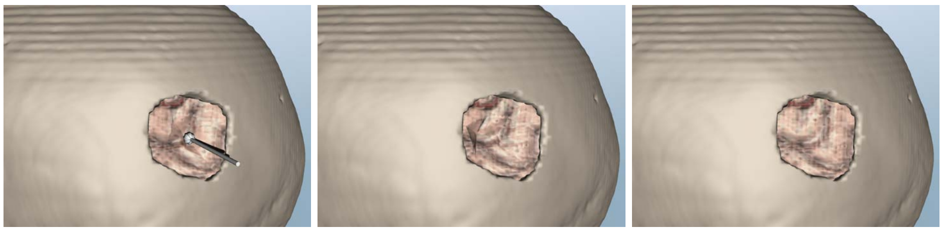


Figure 5.15: Sequence showing the brain interaction.

Thanks to all these optimizations, the resulting simulator (Figure 5.16) runs at interactive times, around 15-20 FPS, with a 1680 × 1050 viewport and achieves the necessary 1000 Hz frequency in the haptic device.

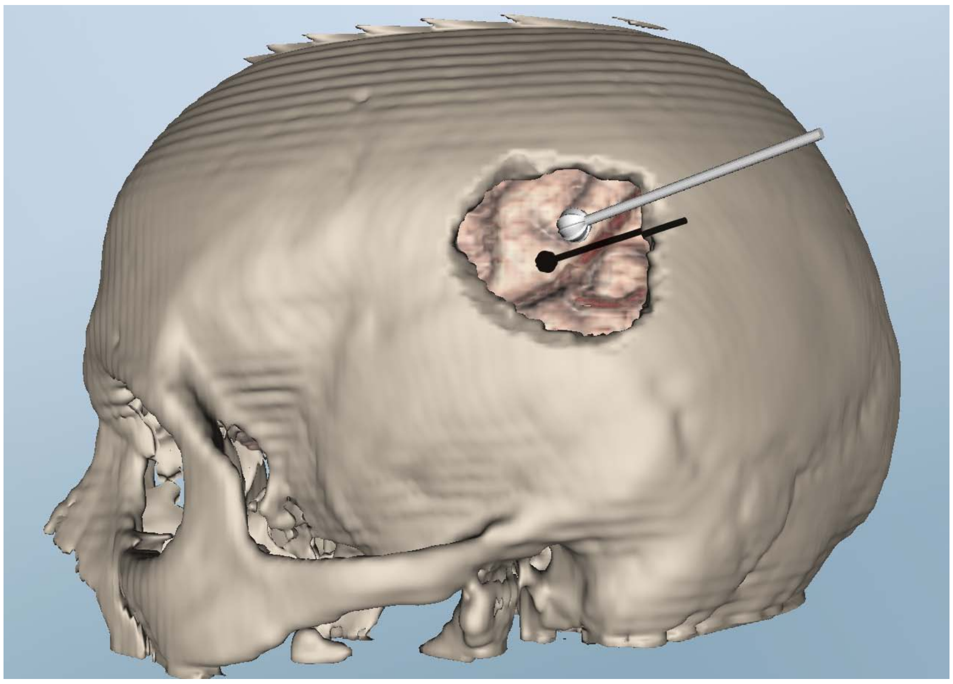


Figure 5.16: Final visualization of the simulator which achieves interactive times with 1680×1050 viewport.

5.5 Discussion

This chapter has presented the developed neurosurgery simulator prototype. The simulator makes use of the methods presented in Chapter 3 for the rigid visualization of the skull, achieving high framerates and quality.

In order to provide the most accurate simulation possible, a finite element simulation has been implemented in the GPU using CUDA. This achieves real time simulation times with high accuracy, which is highly desirable in a medical simulation. Furthermore, using the methods presented in Chapter 4, the results of the simulation can be visualized in real time without resampling the volume. And, using the information given by the study presented in said chapter, the best compromise between quality and performance is achieved.

Adding the haptic device and a precise collision detection and response, a quality neurosurgery simulator prototype is achieved. This simulator is able to simulate the craniotomy procedure and the posterior interaction with the brain accurately.

Part III

Conclusions

Chapter 6

Conclusions and Future Work

We can only see a short distance ahead, but we can see plenty there that needs to be done.
ALAN TURING

6.1 Conclusions

This thesis studies the different volumetric rendering techniques needed for the visualization module of a patient specific surgery simulator. For this purpose the limitations of the state of the art surgery simulators have been studied, along with the problems of the actually available volumetric rendering methods.

In the first stage, the capabilities of the currently available volumetric visualization frameworks have been studied (such as VTK) and, in order to meet the needs of patient specific surgery simulators, improvements on these methods have been developed. These improvements increase the rendering quality maintaining real time rendering speed.

In addition, an special ray traversal method for direct volumetric isosurface rendering has been developed allowing high quality fast isosurface rendering. Thanks to this method, surface extraction is avoided which allows the change of the desired surface without any additional calculations. Additionally, the use of intersection refinement greatly increases the quality with average sampling rate and maintains the quality without major changes if the sampling rate is decreased to speed up the calculations.

These methods have been made available for the public by creating an extended VTK volumetric rendering class with the aforementioned methods. Thanks to this, any developer can use the presented improved framework's features, such as shadow receiving volumes, stochastic jittering or direct isosurface rendering, without any additional effort. Additionally, to further increase the ease of use of style transfer functions, a designing interface has been proposed which allows the user to easily create realistic or illustrative style transfer functions.

The second stage has been focused in the volumetric rendering of deformable volumes, such as soft tissue. For this purpose, a high quality deformable volumetric rendering pipeline has been developed. This pipeline deforms the space of the volume, resulting in an inverse ray deformation and the visualization of the deformed volume. A higher level of continuity for the rays has been proposed, which allows a smooth transition between tetrahedra and increases the quality of the visualization.

In order to correctly render the scene, the volume's illumination must be corrected to reflect the changes in volume space. For this purpose, different illumination correction methods have been presented and studied, i.e. the correction of the structured volume's gradient and the correction of the light and camera direction. Additional improvements have been also studied, such as the use of constant corrections inside each tetrahedra or the use of smoothed corrections between the nodes, which greatly increases the result of the illumination.

The different options in the presented method have been studied so that informed choices and decisions can be made. Furthermore, the different options and all of their combinations have been empirically studied in different usage scenarios, in order to cover the different possible applications.

Finally, using the methods studied in this dissertation, a neurosurgery simulator prototype has been developed. This simulator is within the first stage of a complete neurosurgery simulator, and is capable of simulating the craniotomy procedure (visualized using volumetric rendering of rigid volumes) with a haptic feedback to the user.

It is also able to simulate the interaction with the brain, which is simulated as a deformable body. For this purpose, a physical simulation has been implemented in the GPU so that real time simulation times could be achieved. Thanks to the simulation module was specifically developed for the GPU, not only the simulation speed is increased, but the necessary data transference is kept at a minimum between the simulation and the visualization modules. At the same time, the previously presented methods have been used to visualize the deformed volume while avoiding the resampling of the volume and the use of extracted isosurfaces. With the data from the study, the most efficient options have been selected, thus ensuring interactive visualization while maintaining the highest quality possible.

6.2 Future research lines

Although the presented methods allow the use of high quality visualization in real time, several research lines that could further improve the quality and performance are still open:

- As raycasting is a highly flexible volume rendering method, most rendering modes and effects can be directly used even if the ray is deformed, as it has been done with the stochastic jittering. However, other methods exist that may need further modifications or optimizations to use them with deformed rays. For example, some methods to calculate scattering or ambient occlusion use additional rays exiting from the sample point.

- The deformable rendering method only uses tetrahedra as the primitives for the rendering. However, many simulations use mixed representations, often hexahedra and tetrahedra. The generalization of the rendering method would allow the direct use of other primitives, as well as potentially allow further improvements using the additional information.

- The simulator prototype presented in this dissertation should be validated by the users. This users must be both expert and trainees in order to better asses the achieved knowledge transfer and the realism of the procedures, tools and interactions in the simulator.

- In order to continue the progress towards a full neurosurgery simulators many task must be included in the simulator. One of such task is the cutting and retraction of tissue, for which the simulation must be adapted. Additionally, the tumour resection or extraction tasks need to be correctly visualized. It would be highly desirable to maintain the volumetric rendering by creating a method capable of visualizing these methods using the volume.

- The creation of the tetrahedra depends heavily in the segmentation of the brain in the MRI volume. However, nowadays segmentation of the brain is a task that cannot be performed in a completely automatic way. The research and development of an automatic brain segmentation algorithm would allow the use of a fully automatic tetrahedral mesh creation procedure, which would be a very desirable feature.

- The used tetrahedral mesh assumes constant properties through the whole brain, which is known not to be accurate. A process to create a more exact tetrahedral mesh is necessary as, in order to progress towards a full neurosurgery simulator, the properties of the different parts of the brain have to be taken into account in the simulation. Additionally, this information could be used to enhance the visualization of the underlying volume.

- The developed CUDA implementation for the FEM simulation can be expanded and improved. The used data structure can be reorganized so that only the necessary memory is allocated, which would greatly decrease the impact on the memory.

Part IV

Appendices

Appendix A

Implementation of Direct Volumetric Isosurface Rendering and Shadow Mapping for VTK

This appendix describes the extension of a Visualization Toolkit (VTK)[1] mapper for the rendering of refined direct volumetric isosurfaces. Additionally, it also adds shadow mapping for volume rendering in both rendering modes, full volume rendering (FVR) and direct isosurface rendering.

Refined direct volumetric isosurface rendering allows faster rendering times when visualizing isosurfaces from volume data. Also, the resulting rendering provides a much higher quality rendering. However, VTK does not provide the necessary classes to perform this type of rendering. Additionally, VTK's pipeline does not allow the volumes to receive shadows, which is a highly desirable feature when developing simulators. This two limitations are tackled by the presented class, while maintaining full compatibility with VTK. The implementation details and necessary changes to seamlessly include these features in VTK are documented in this appendix.

[1] `http://www.vtk.org`

A.1 Introduction

Volumetric rendering is a key technique in scientific and medical visualization. This rendering method enables the direct use of volumes as data for the rendering pipeline, avoiding the need of preprocessing and the possible loss of data and accuracy it could bring. Volume rendering allows the inspection of the whole volume at once or, at least, in an interactive and intuitive mode. This makes it possible to inspect the results of a simulation as a whole, or see patients' internal organs without inspecting an MRI scan slice by slice.

But sometimes the user might want to inspect certain specific values of the volume, such as the organs or bones from a CT scan or a shock wave in a simulation. Usually, in order to visualize said isosurfaces, an isosurface extraction method such as marching cubes would be used and a regular polygonal rendering would be performed with the result. Another method would be using a transfer function set to be completely opaque at the desired isosurface, but this method has the high computational cost of volume rendering without taking advantage of its strong points, i.e., the rendering of semitransparent volumes.

Another method for the visualization of isosurfaces is volumetric isosurface rendering. With this the extraction method is skipped and the volume is directly rendered by setting an isovalue. In addition, by using an intersection refinement method the step size can be increased with little or no quality loss. In this way the method offers the flexibility and quality of volume rendering without its high computational cost.

The Visualization ToolKit (VTK) offers a wide range of the said methods, but the lack of a volumetric isosurface rendering poses a problem in certain situations, such as a medical visualization using CTs. Moreover, VTK does not allow the volumes to receive the shadows cast by polygonal surfaces, a much needed feature when dealing with user's depth perception of such polygonal actors.

So, in order to fill this gap, a a extension of the vtkOpenGLGPUVolumeRayCastMapper is presented (called vtkOpenGLGPUVolumeRayCastMapper2 to allow compatibility and simultaneous use) which allows the use of a volumetric isosurface mode with intersection refinement and receiving shadows from polygonal actors without modifying the shadow rendering pipeline.

A.2 Changes on the design

The proposed changes do not require new classes by themselves, but in order to allow a easier usability and avoid potential problems with the different versions of the vtkOpenGLGPUVolumeRayCastMapper and vtkGPUVolumeRayCastMapper two new classes have been created that are basically an extension of the originals: vtkOpenGLGPUVolumeRayCastMapper2 and vtkGPUVolumeRayCastMapper2. Both classes have been created in order to maintain the usual distribution of code in the original classes and to maintain the same design as shown in Figure A.1.

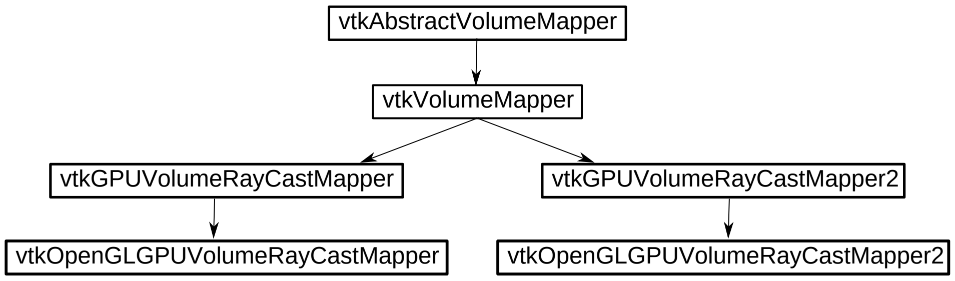

vtkAbstractVolumeMapper
vtkVolumeMapper
vtkGPUVolumeRayCastMapper
vtkGPUVolumeRayCastMapper2
vtkOpenGLGPUVolumeRayCastMapper
vtkOpenGLGPUVolumeRayCastMapper2

Figure A.1: The resulting design with the new classes.

To summarize the additions and changes done to the classes:

- A new rendering mode has been added, the volumetric isosurface mode. For this a new GLSL shader was created, which implements the isosurface ray traversal mode with the refinement. Additionally, a variable to set the isovalue has been added along with the methods to access and modify it.

- The shadow receiving mode has been added, which is controlled in the same way it is controlled for polygonal objects. Three new GLSL shaders have been created too, implementing different shadow modes: no shadows, hard shadows and soft shadows. Several variables and methods to access said variables have been added.

- Additionally, the code for stochastic jittering has been added as well as several variables and methods to control it. Two GLSL shaders have also been created, stochastic jittering enabled and disabled. Note that the methods and variables for the stochastic jittering maintain the nomenclature from VTK, using noise instead of stochastic jittering.

A.2.1 Volumetric isosurface rendering

To include refined isosurface rendering to the original VTK volume mapper, an additional ray traversal mode has to be added. This mode will traverse the entire ray until the sample found is higher than the isovalue. With this sample position an intersection refinement will be performed by iterating using the Eqaution A.1.

$$X_{new} = (X_{next} - X_{prev}))\,\frac{isovalue - f_{prev}}{f_{next} - f_{prev}} + X_{prev} \tag{A.1}$$

With the new intersection point, the value of the new point is read. That new value is then compared to the isovalue, if its bigger then X_{new} and the new value are assigned to X_{prev} and f_{prev} respectively. Otherwise, X_{new} and the new value are assigned to X_{next} and f_{next}. The Algorithm A.1, presents the raytraversal for isosurface rendering with intersection refined.

This rendering allows a increase in performance while maintaining a high quality visualization, as it has been shown in Section 3.

A.2.2 Shadows

Shadows are a key feature to enhance both realism and depth perception as shadows help the user to locate the position of an object, i.e., a tool in a simulator. Thus they are a very desirable feature in many visualizations.

VTK includes a method to create a pipeline with shadows, using vtkShadowMapPass and vtkShadowMapBakerPass. In this pipeline, the polygonal actors are rendered as usual and after this the shadows are rendered onto the previous rendering.

When adding shadows to the volume, if the usual pipeline were to be used, the volume would have to been rendering twice, with the evident result

Algorithm A.1 Ray traversal for direct volumetric isosurface rendering with intersection refinement.

```
1: function Ray traversal for direct isosurface rendering(...)
2:     while $InsideTheVolume(CurrentPoint)$ do
3:         $Value = ValueFromPoint(CurrentPoint)$
4:         if $Value \geq Isovalue$ then
5:             for $i = 0 \rightarrow 10$ do
6:                 $Fraction = \frac{Isovalue - PreviousValue}{CurrentValue - PreviousValue}$
7:                 $Diff = (CurrentPoint - PreviousPoint)$
8:                 $NewPoint = Diff * Fraction + PreviousPoint$
9:                 $NewValue = ValueFromPoint(NewPoint)$
10:                if $NewValue < IsoValue$ then
11:                    $PreviousValue = NewValue$
12:                    $PreviousPoint = NewPoint$
13:                else
14:                    $CurrentValue = NewValue$
15:                    $CurrentPoint = NewPoint$
16:                end if
17:            end for
18:            $Shade(IsoValue, NewPoint)$
19:            $return$
20:        end if
21:        $CurrentPoint = NextPoint$
22:    end while
23: end function
```

in performance loss. In order to avoid this loss, a more common approach was taken. Instead of rendering the shadows separately, the shadowing is checked when the ray traversal is performed. As a result, shadows from the polygons can be cast onto the volume with little additional computational cost, resulting in an improved depth perception as it has been already shown in Section 3.

A.3 Usage examples

Along with the source code of the new classes, a example application has been prepared, where the examples presented below are demonstrated. The minimum version of VTK to use these classes is 5.8, as the shadows pipeline was changed from 5.6 to 5.8

A.3.1 Isosurface mode usage

The usage method for this rendering mode slightly differs from the usual choice of blending mode in volumetric rendering. The reasoning behind this difference is that to maintain the usual pipeline, further changes should be made to other classes, and it was deemed undesirable at the current development stage. If in the future these classes are merged into VTK the usage method will change to the usual VTK mode. An example of how to set the isosurface to `value`:

```
vtkGPUVolumeRayCastMapper2 * mapper =
                 vtkGPUVolumeRayCastMapper2::New();
...
mapper->SetIsovalue(value);
mapper->SetBlendModeIsosurface(true);
```

When `BlendModeIsosurface` is set to true, the blending mode chosen by the usual means will have no effect, and it will be always rendered as isosurface. To use any other blending mode, `BlendModeIsosurface` has to be set to false. Note that the original blending mode is not changed by changing `BlendModeIsosurface` so just setting it to false will force the mapper to fall back to the previous blending mode.

A.3.2 Receiving shadows

Enabling the shadows for the volume rendering is performed as usual in VTK. A rendering pipeline including `vtkShadowMapPass` and `vtkShadowMapBakerPass` must be set up, and the polygonal objects properties for the shadows are set in the original way. In order to enable the volume to receive shadows, a `vtkShadowMapBakerPass::RECEIVER` must be set (regardless of to 0 or 1). If in addition to shadows, soft shadows are wanted they must be activated by setting `SoftShadows` to true. The offset of the soft shadow is controlled with `SoftShadowsOffset` and it represents the offset in pixels. Note that although it represents pixels, it does not need to be integer as linear interpolation is used for the shadow texture.

```
vtkGPUVolumeRayCastMapper2 * mapper =
                 vtkGPUVolumeRayCastMapper2::New();
...
vtkVolume * volume = vtkVolume::New;
volume->SetMapper(mapper);

vtkInformation * info = vtkInformation::New();
info->Set(vtkShadowMapBakerPass::RECEIVER(), 0);

volume->SetPropertyKeys(info);

//For soft shadows:
mapper->SetSoftShadows(true);
mapper->SetSoftShadowsOffset(1.0);
```

A.3.3 Using stochastic jittering (Noise)

To enable the use of stochastic jittering `Noise` must be set to true. Additionally, the noise texture size and the maximum inserted stochastic jittering can be controlled through `NoiseTextureSize` and `MaxNoiseFactor` respectively.

```
vtkGPUVolumeRayCastMapper2 * mapper =
                 vtkGPUVolumeRayCastMapper2::New();
...
mapper->SetNoise(true);
mapper->SetMaxNoiseFactor(0.75);
mapper->SetNoiseTextureSize(64);
```

A.4 Conclusions and Future Work

The presented mapper allows the use of direct volumetric isosurface rendering with intersection refinement efficiently, allowing the visualization of isosurfaces with great quality and performance. In addition, the volume has been enabled to receive shadows from VTKs pipeline while avoiding the need for costly additional volumetric render passes. These two features increase the possibilities offered by VTK in terms of immersion. Moreover, the activation of the stochastic jittering further increases the visualization quality.

The presented mapper for VTK adds these highly desirable features, while maintaining all the existent functionality that VTK offers. This will allow any non expert developer the use of these methods to increase the realism and immersion of their applications easily.

The future work will be focused in these points:

- Right now the mapper only uses one light for the illumination (like the original mapper), having all or more light into account may be a desirable expansion
- The shadows are only received from one light, the same as the lighting, and it may be desirable to expand this along the number of lights
- Although volumes can receive shadows, they do not cast shadows. It is a desirable feature to expand this mapper and the VTK passes for the shadows in order to allow the use shadows from the volumes.

Appendix B

Generated Publications

Journals

Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Study of ray casting techniques for the visualization of deformable volumes". *IEEE Transaction on Visualization and Computer Graphics*, 2013. Accepted with Major Revision.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Simulation and visualization techniques for a surgery simulator of skull drilling and brain interaction tasks". *IEEE Computer Graphics and Applications*, 2013. Accepted.

Herrera, I., Buchart, C., and Borro, D. "Adding refined isosurface rendering and shadow mapping to vtkgpuvolumeraycastmapper". *VTK Journal (Midas Journal)*, October, 2012.

Conferences

Herrera, I., Buchart, C., and Borro, D. "Preserving coherent illumination in style transfer functions for volume rendering". In *Information Visualisation (IV), 2010 14th International Conference DOI - 10.1109/IV.2010.16*, pp. 43–47. 2010.

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. December 14, 2010.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data". In *Proceedings of the XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011)*, pp. 423–426. 2011. Cáceres, Spain. November 16-18, 2011.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. "Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization". In *Proceedings of the XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012).* San Sebastián, Spain. November 19-21, 2012.

References

Bathe, K.-J. *Finite element procedures*. Prentice Hall. 1996.

Bavoil, L., Sainz, M., and Dimitrov, R. “Image-space horizon-based ambient occlusion”. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH ’08, pp. 22:1–22:1. New York, NY, USA. 2008.

Bernardon, F. F., Pagot, C. A., Comba, J. L. D., and Silva, C. T. “Gpu-based tiled ray casting using depth peeling”. *Journal of Graphics, GPU, and Game Tools*, Vol. 11, N. 4, pp. 1–16. 2006.

Bétrancourt, M. and Tversky, B. “Effect of computer animation on users’ performance: A review”. *Le travail humain*, Vol. 63, N. 4, pp. 311–329. 2000.

Bhanirantka, P. and Demange, Y. “Opengl volumizer: a toolkit for high quality volume rendering of large data sets”. In *Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH Symposium on*, pp. 45 – 53. oct., 2002.

Birkeland, A. and Viola, I. “View-dependent peel-away visualization for volumetric data”. In *Proceedings of the 25th Spring Conference on Computer Graphics*, SCCG ’09, pp. 121–128. New York, NY, USA. 2009.

Bruckner, S. and Groeller, M. E. “Style transfer functions for illustrative volume rendering”. *COMPUTER GRAPHICS FORUM*, Vol. 26, N. 3, Sp. Iss. SI, pp. 715–724. 2007. 28th Annual Conference of the European-Association-for-Computer-Graphics (EUROGRAPHICS 2007), Prague, CZECH REPUBLIC, SEP 03-07, 2007.

Buchart, C., San Vicente, G., Amundarain, A., and Borro, D. “Hybrid visualization for maxillofacial surgery planning and simulation”. In *INFORMATION VISUALIZATION, IV 2009, PROCEEDINGS*, pp. 266–273. 2009. 4th Information Visualization Conference, Barcelona, SPAIN, JUL 15-17, 2009.

Cabral, B., Cam, N., and Foran, J. “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware”. In *Proceedings of the 1994 symposium on Volume visualization*, VVS ’94, pp. 91–98. New York, NY, USA. 1994.

Chen, H., Hesser, J., and Männer, R. “Fast volume deformation using inverse-ray-deformation and ffd”. In *International Conference Graphicon*. 2001.

Chen, M., Silver, D., Winter, A. S., Singh, V., and Cornea, N. “Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation”. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, VG ’03, pp. 35–44. New York, NY, USA. 2003.

Clarke, D. B., D’Arcy, R. C., Delorme, S., Laroche, D., Godin, G., Hajra, S. G., Brooks, R., and DiRaddo, R. “Virtual reality simulator: Demonstrated use in neurosurgical oncology”. *Surgical Innovation*, Vol. 20, N. 2, pp. 190–197. 2013.

Correa, C. and Ma, K.-L. “Visibility histograms and visibility-driven transfer functions”. *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 17, N. 2, pp. 192–204. 2011.

Correa, C., Silver, D., and Chen, M. “Illustrative deformation for data exploration”. *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 13, N. 6, pp. 1320 –1327. nov.-dec., 2007.

Correa, C. D., Silver, D., and Chen, M. “Constrained illustrative volume deformation”. *Computers & Graphics*, Vol. 34, N. 4, pp. 370 – 377. 2010. Procedural Methods in Computer Graphics Illustrative Visualization.

Delorme, S., Laroche, D., DiRaddo, R., and F. Del Maestro, R. “Neurotouch: A physics-based virtual simulator for cranial microneurosurgery training”. *Neurosurgery*, Vol. 71, pp. –. 2012.

Díaz, J., Váquez, P.-P., Navazo, I., and Duguet, F. “Real-time ambient occlusion and halos with summed area tables”. *Computers & Graphics*, Vol. 34, N. 4, pp. 337 – 350. 2010.

Echegaray, G. *Rigid and deformable collision handling for a haptic neurosurgery simulator*. PhD thesis, University of Navarra. 2012.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. “Towards a multimodal neurosurgery simulator: Brain haptic physical simulation and visualization”. In *Proceedings of the XXX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2012)*. San Sebastián, Spain. November 19-21, 2012.

Echegaray, G., Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. “Simulation and visualization techniques for a surgery simulator of skull drilling and brain interaction tasks”. *IEEE Computer Graphics and Applications*, 2013. Accepted.

Echegaray, G., Herrera, I., Buchart, C., and Borro, D. “Towards a multimodal neurosurgery simulator: Drilling simulation and visualization using real patient data”. In *Proceedings of the XXIX Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2011)*, pp. 423–426. 2011. Cáceres, Spain. November 16-18, 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. December 14, 2010.

Engel, K., Hadwiger, M., Kniss, J., Lefhon, A., Rezk-Salama, C., and Weiskopf, D. *Real-time volume graphics*. AK Peters. 2006.

Gagvani, N. and Silver, D. “Animating volumetric models”. *Graphical Models*, Vol. 63, N. 6, pp. 443 – 458. 2001.

Georgii, J. and Westermann, R. “A generic and scalable pipeline for gpu tetrahedral grid rendering”. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 12, N. 5, pp. 1345–1352. SEP-OCT, 2006. IEEE Visualization Conference (Vis

2006)/IEEE Symposium on Information Visualization (InfoVis 2006), Baltimore, MD, OCT 29-NOV 03, 2006.

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.

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.

Goldstone, R. L. and Son, J. Y. “The transfer of scientific principles using concrete and idealized simulations”. *Journal of the Learning Sciences*, Vol. 14, N. 1, pp. 69–110. 2005.

Grau, S. and Puig, A. “An adaptive cutaway with volume context preservation”. In Bebis, G., Boyle, R., Parvin, B., Koracin, D., Kuno, Y., Wang, J., Pajarola, R., Lindstrom, P., Hinkenjann, A., EncarnaÃ§Ã£o, M., Silva, C., and Coming, D., editors, *Advances in Visual Computing*, volume 5876 of *Lecture Notes in Computer Science*, pp. 847–856. Springer Berlin Heidelberg. 2009.

Hadwiger, M., Ljung, P., Salama, C. R., and Ropinski, T. “Advanced illumination techniques for gpu volume raycasting”. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia ’08, pp. 1:1–1:166. New York, NY, USA. 2008.

Haines, E. *Graphics gems ii*, chapter Fast Ray–Convex Polyhedron Intersection, pp. 247–250. Academic Press. 1991.

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

Herghelegiu, P. and Manta, V. “Volume illumination based on lit sphere maps”. Technical Report Tome LIV (LVIII), The ”Gheorghe Asachi” Technical University of Iasi. 2008.

Herrera, I., Aguinaga, I., Buchart, C., and Borro, D. “Study of ray casting techniques for the visualization of deformable volumes”. *IEEE Transaction on Visualization and Computer Graphics*, 2013. Accepted with Major Revision.

Herrera, I., Buchart, C., and Borro, D. "Preserving coherent illumination in style transfer functions for volume rendering". In *Information Visualisation (IV), 2010 14th International Conference DOI - 10.1109/IV.2010.16*, pp. 43–47. 2010.

Herrera, I., Buchart, C., and Borro, D. "Adding refined isosurface rendering and shadow mapping to vtkgpuvolumeraycastmapper". *VTK Journal (Midas Journal)*, October, 2012.

Hounsfield, G. N. "Apparatus for examining bodies by means of penetrating radiation". 1976. US Patent 3,946,234.

Irving, G., Teran, J., and Fedkiv, R. "Tetrahedral and hexahedral invertible finite elements". *Graphical Models*, Vol. 68, N. 2, pp. 66–89. 2004.

Ji, G., Shen, H.-W., and Gao, J. "Interactive exploration of remote isosurfaces with point-based non-photorealistic rendering". In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pp. 25–32. 2008.

Jones, K. and McGee, J. "Opengl volumizer programmer's guide". 2005.

JS, D. and S, A. "A computer-controlled patient simulator". *JAMA*, Vol. 208, N. 3, pp. 504–508. 1969.

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.

Kruger, J. and Westermann, R. "Acceleration techniques for gpu-based volume rendering". In *Proceedings IEEE Visualization 2003.* 2003.

Kurzion, Y. and Yagel, R. "Interactive space deformation with hardware-assisted rendering". *Computer Graphics and Applications, IEEE*, Vol. 17, N. 5, pp. 66 –77. September, 1997.

Lacroute, P. and Levoy, M. "Fast volume rendering using a shear-warp factorization of the viewing transformation". In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pp. 451–458. New York, NY, USA. 1994.

Lathen, G., Lindholm, S., Lenz, R., Persson, A., and Borga, M. "Automatic tuning of spatially varying transfer functions for blood vessel visualization". *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 18, N. 12, pp. 2345–2354. 2012.

Lauterbur, P. C., Others, and Others. "Image formation by induced local interactions: examples employing nuclear magnetic resonance". *Nature*, Vol. 242, N. 5394, pp. 190–191. 1973.

Lehmann, T., Oberschelp, W., Pelikan, E., and Repges, R. *Bildverarbeitung für die medizin*, volume 1. Springer Berlin. 1997.

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.

Levoy, M. "Display of surfaces from volume data". *Computer Graphics and Applications, IEEE*, Vol. 8, N. 3, pp. 29–37. 1988.

Lorensen, W. E. and Cline, H. E. "Marching cubes: A high resolution 3d surface construction algorithm". In *ACM Siggraph Computer Graphics*, volume 21, pp. 163–169. 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.

Maciejewski, R., Woo, I., Chen, W., and Ebert, D. S. "Structuring feature space: A non-parametric method for volumetric transfer function generation". *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 15, N. 6, pp. 1473–1480. NOV-DEC, 2009. IEEE Information Visualization Conference/IEEE Visualization Conference, Atlantic City, NJ, OCT 11, 2009.

Mansfield, P. "Multi-planar image formation using nmr spin echoes". *Journal of Physics C: Solid State Physics*, Vol. 10, N. 3, p. L55. 1977.

Marmitt, G. and Slusallek, P. "Fast ray traversal of tetrahedral and hexahedral meshes for direct volume rendering". 2006.

McGuffin, M. J., Tancau, L., and Balakrishnan, R. “Using deformations for browsing volumetric data”. In *IN PROCEEDINGS OF IEEE VISUALIZATION 2003*, pp. 401–408. 2003.

Monclús, E., Díaz, J., Navazo, I., and Vázquez, P.-P. “The virtual magic lantern: an interaction metaphor for enhanced medical data inspection”. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, VRST ’09, pp. 119–122. New York, NY, USA. 2009.

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.

Muigg, P., Hadwiger, M., Doleisch, H., and Groeller, E. “Interactive volume visualization of general polyhedral grids”. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 17, N. 12, pp. 2115–2124. DEC, 2011. IEEE Visualization Conference (Vis)/IEEE Information Visualization Conference (InfoVis), Providence, RI, OCT 23-28, 2011.

Muigg, P., Hadwiger, M., Doleisch, H., and Hauser, H. “Scalable hybrid unstructured and structured grid raycasting”. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 13, N. 6, pp. 1592–1599. NOV-DEC, 2007. IEEE Visualization Conference (Vis 2007)/IEEE Information Visualization Conference (InfoVis 2007), Sacramento, CA, OCT 28-NOV 01, 2007.

Nagy, Z., Schneider, J., and Westermann, R. “Interactive volume illustration.” In *VMV*, pp. 497–504. 2002.

Nakao, M., Hung, K. W. C., Yano, S., Yoshimura, K., and Minato, K. “Adaptive proxy geometry for direct volume manipulation”. In North, S., Shen, H., and Vanwijk, J., editors, *IEEE PACIFIC VISUALIZATION SYMPOSIUM 2010*, pp. 161–168. 2010. IEEE Pacific Visualization Symposium, Taipei, TAIWAN, MAR 02-05, 2010.

Petersik, A., Pflesser, B., Tiede, U., Höhne, K., and Leuwer, R. "Realistic haptic volume interaction for petrous bone surgery simulation". *CARS*, Vol. 1, pp. 252–257. 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.

Platis, N. and Theoharis, T. "Fast ray-tetrahedron intersection using plücker coordinates". *journal of graphics, gpu, and game tools*, Vol. 8, N. 4, pp. 37–48. 2003.

Rautek, P., Bruckner, S., and Groller, M. E. "Semantic layers for illustrative volume rendering". *Ieee Transactions On Visualization And Computer Graphics*, Vol. 13, N. 6, pp. 1336–1343. 2007.

Rezk-Salama, C., Scheuering, M., Soza, G., and Greiner, G. "Fast volumetric deformation on general purpose hardware". In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pp. 17–24. New York, NY, USA. 2001.

Rhee, T., Lewis, J., Neumann, U., and Nayak, K. "Scan-based volume animation driven by locally adaptive articulated registrations". *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 17, N. 3, pp. 368–379. 2011.

Roettger, S., Guthe, S., Weiskopf, D., Ertl, T., and Strasser, W. "Smart hardware-accelerated volume rendering". In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pp. 231–238. Aire-la-Ville, Switzerland, Switzerland. 2003.

Ruiz, M., Bardera, A., Boada, I., Viola, I., Feixas, M., and Sbert, M. "Automatic transfer functions based on informational divergence". *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 17, N. 12, pp. 1932–1941. 2011.

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

Sorensen, M. S., Mosegaard, J., and Trier, P. "The visible ear simulator: a public pc application for gpu-accelerated haptic 3d simulation of ear surgery based on the visible ear data". *Otology & Neurotology*, Vol. 30, N. 4, pp. 484–487. 2009.

Stolfi, J. "Oriented projective geometry". In *Proceedings of the third annual symposium on Computational geometry*, pp. 76–85. 1987.

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.

Teran, J., Blemker, S., Hing, V. N. T., and Fedkiw, R. "Finite volume methods for the simulation of skeletal muscle". In *Eurographics/SIGGRAPH Symposium on Computer Animation*. 2003.

Thomas, M. P. "The role of simulation in the development of technical competence during surgical training: a literature review". *Int J Med Educ*, Vol. 4, pp. 48–58. 2013.

Wang, L., Zhao, X., and Kaufman, A. "Modified dendrogram of attribute space for multidimensional transfer function design". *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 18, N. 1, pp. 121–131. 2012.

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.

Weiler, M., Kraus, M., Merz, M., and Ertl, T. "Hardware-based ray casting for tetrahedral meshes". In *Visualization, 2003. VIS 2003. IEEE*, pp. 333 –340. oct., 2003.

Weiler, M., Mallon, P., Kraus, M., and Ertl, T. "Texture-encoded tetrahedral strips". In *Volume Visualization and Graphics, 2004 IEEE Symposium on*, pp. 71 – 78. oct., 2004.

Westermann, R. and Rezk-Salama, C. "Real-time volume deformations". *Computer Graphics Forum*, Vol. 20, N. 3, pp. 443–451. 2001.

Westover, L. A. *Splatting: a parallel, feed-forward volume rendering algorithm*. PhD thesis, University of North Carolina at Chapel Hill. 1991.

Wittenbrink, C., Malzbender, T., and Goss, M. "Opacity-weighted color interpolation for volume sampling". In *Volume Visualization, 1998. IEEE Symposium on*, pp. 135–142. 1998.

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.

Zhou, J. and Takatsuka, M. "Automatic transfer function generation using contour tree controlled residue flow model and color harmonics". *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 15, N. 6, pp. 1481–1488. NOV-DEC, 2009. IEEE Information Visualization Conference/IEEE Visualization Conference, Atlantic City, NJ, OCT 11, 2009.

Zhu, Q.-h., Chen, Y., and Kaufman, A. "Real-time biomechanically-based muscle volume deformation using fem". *Computer Graphics Forum*, Vol. 17, N. 3, pp. 275–284. 1998.