Imperial College London

Department of Computing

Haptic Deformable Shapes Using Open Source Libraries

by

Alejandro Granados

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London

September 2008

“Colour is the keyboard,

the eyes are the harmonies,

the soul is the piano with many strings.

The artist is the hand that plays,

touching one key or another,

to cause vibrations in the soul.”

Wassily Kandinsky
1866-1944

Acknowledgements

This dissertation report is dedicated to my parents Blanca and Alejandro and to my sister Verónica who are indeed my entire life.

Especial thanks to Katia Castañeda who I admire and love and whose closeness and adoration have always been enriching my life.

I absolutely appreciate the help, advice and time of Dr. Fernando Bello and Dr. Pierre-Frederic Villard whose work, ideas, comments and suggestions have been both professional and inspiring.

To my godchildren Alejandro, Daniela and Andrea and to my niece Ivonne and nephew Manuel who are a source of joy in my life in addition to my family Blanca and Erik, and Gerardo and Verónica to whom I am delighted to be their brother.

Many thanks to my friends Rene Barrera, Edgar Flores and Miguel Toledo who I respect and admire and with whom I have spent the greatest moments together as friends and as professionals.

Thanks to Carlos Boullosa for his worthy friendship and encouragement.

To my friends Eleni, Chara, Kyriacos, Jhyadi, Josiah, David, Antonis, Rafal, Lee and Javid to whom I feel fortunate to meet in London.

To my English friend Henry Atterbury for all his invaluable stories about London and his inspiring personality about classical music.

Abstract

Surgical simulation in medical applications bridges the gap between training and performing the real intervention of patients. In this work, both haptic rendering and deformation modelling are emphasized and represent two crucial areas of any surgical simulation.

Haptics substantially increases the quality of interaction by adding the sense of touch. Physiological and psychophysical foundations are present in the design of available haptic interfaces. Haptic rendering consists of collision-detection, force-response and control algorithms which are processed in high-frequency servo rates.

To date, research has looked into better ways for approximating a physically-based modelling due to the nature of deformable soft tissue objects found in human organs, especially in multi-resolution approaches. Heuristic approaches and continuum-based mechanic approaches are described.

The advent of appealing open source software frameworks, such as SOFA and H3D, presents new challenges that will allow a rapid integration platform for deformable objects and haptic rendering in research.

Finally, a framework for haptic deformable shapes is presented combining haptic rendering, deformation modelling and open source tools. A virtual human liver is used for the evaluation and specific procedures such as needle insertion are examined.

Table of Contents

Introduction

Motivations

Surgical simulation in medical applications bridges the gap between training and performing the real intervention of patients. Haptic features are inherent in a virtual environment with these characteristics. Thus, haptics in training plays an important role for the framework as well as the deformation model used.

The multi-disciplinary aspect of this field requires the integration of state-of-the-art solutions in a diverse range of areas. As an example, advanced techniques for FEM approximation together with available multi-resolution methods, and local refinement make easier to simulate an interaction environment in a single-processor computing system. The use of GPUs might even help for real-time applications. Additionally, haptic devices, such as NOVINT®'s Falcon, are an important leverage for this project due to its availability as a consumer-based product.

Finally, the availability and flexibility of recent open source libraries for the simulation engine and the haptic rendering bring more possibilities to this research.

Objectives

"The aim of this project is to use the Open Source libraries H3D and SOFA to build a framework for haptic deformable shapes that will allow the interaction with objects that will deform according to an underlying mathematical model. Such system will facilitate future development of surgical simulators by offering a common development platform."

Structure

This document comprises five chapters called background, design, implementation, analysis and evaluation.

The background chapter is divided mainly in three general sections. In the first part haptic rendering is described. Design guidelines are based on human foundations and the importance of multi-modal environments using vision and haptics is emphasised. Haptic rendering architecture is depicted along with some collision-detection, force-response and control algorithms. A brief introduction to haptic interfaces is included and key features of nonlinear behaviour are considered at the end of this section. In the second part, nonlinear deformation models are described with especial interest in those representing physical characteristics of human soft tissue. Deformable approaches are classified in three main groups: heuristic, continuum-based and multi-resolution models. Time discretisation effects are considered and an analysis of remeshing techniques is discussed. Finally, in the third section, open source libraries are analysed. Firstly, SOFA framework is described as it will be used for the simulation engine. H3D is described shortly thereafter as the haptic rendering engine. To finish, FLTK will be used as the GUI platform.

The design chapter includes a summary of the project from different points of view. It serves as a quick reference to see the overall of the implemented framework. It covers information about the hardware platform used for the development and highlights the key aspects in the software architecture proposed for the Haptics Open Framework. By means of UML sequence diagram, this chapter explains how the open source tools were integrated as an overall. It also describes what algorithms and features are used regarding to haptic rendering and deformation modelling.

The implementation chapter provides more detailed information about the software architecture and how the open source tools have been integrated referring to implemented classes. It also explains briefly the GUI components of the framework. In the last part of this chapter, a specific application is described, modelled and implemented for needle insertion procedures.

The analysis chapter links some implemented features in SOFA and H3D to the existing literature in a way to support the decisions taken into the framework and to provide a fundament before performing tests and confirming results in the following chapter.

Finally the evaluation chapter describe specific test scenarios and their respective results as a proof of the proposed framework. Shortly thereafter, some conclusions and future work are manifested.

Background

Haptic Rendering

Haptics

During the last two decades the sense of touch has become an important research topic due to its importance in human-computer interaction systems. The understanding of sensory, perceptual and cognitive abilities in humans has become a turning point in the design and implementation of haptic-enabled systems. In consequence, disciplines ranging from robotics and telerobotics to psychophysics, cognitive science and neuroscience have presented an increased activity in this field.

'Haptic', from the Greek *haptesthai*, pertains to the sense of touch and refers to the variety of possible human and robot interactions with real, remote or virtual objects. The process of conveying object properties as sensory stimuli to a user is called *haptic rendering*, as described by Salisbury et al [2004].

Surgical simulation and medical training along with the game industry have been the primary application areas for haptics. In addition, haptics represents an alternative modality to sound and vision for visually impaired people and becomes a supplement to military applications. Decorating and sculpting in virtual environments are common example applications in virtual arts by interacting with virtual objects in real time. Museums benefit from this principle by allowing visitors to interact with artefacts in 3D. A more detailed explanation of these and more applications can be found in McLaughlin [2005] and Kim [2004].

Human Foundations

Physiological and *psychophysical* knowledge about the sense of touch helps as a guideline for designing haptic interfaces as proposed by Hale and Stanney [2004]. A *multimodal environment* is inherent in human beings and both perception and cognition play an important role while interacting with this environment. Touch benefits from an independent sensory bidirectional channel that speeds reaction time and reduce hand-eye coordination errors.

Physiologically, there are two different types of haptic stimulation: *tactile* and *kinaesthetic*. This classification depends mainly on the location of the receptors. On the one hand, tactile mechanoreceptors are located in the skin and detect touch and pressure. There are four distinct receptors in glabrous (hairless) skin which include Meissner and Pacinian corpuscles, Merkel disks and Ruffini endings. Hands are a good example of glabrous skin which is most effective for detailed tactile information. Additionally, Pacinian corpuscles and Ruffini endings are present in hairy skin along with a third receptor called hair follicles.

On the other hand, kinaesthetic receptors can be found in joints, tendons and muscles and help sense how fast and in which direction the limbs are moving. Ruffini and Golgi endings are located in or near joints and are considered protective receptors. Golgi tendon organ receptors are located in tendons and sense active positioning. Located in muscles, muscle spindles receptors sense limb weight and consciously provide awareness of body movement.

Psychophysics is a branch of psychology and studies the relationship between objects and the perception of its geometric and physical properties as described by Choi and Tan [2004]. Psychophysical experiments have been widely used by researchers to understand the ranges, thresholds and stimulation of particular senses and evaluate multi-receptor perception and its limitations. The results acquired are of extreme importance in the design of a haptic interface.

Hale and Stanney [2004] also consider cross-modal effects as a design guideline. Neurologically, it has been proved that vision and touch are strongly linked together in dynamic contact information. The benefits of this analysis suggest the incorporation of multimodal stimuli into *priming* and *cueing* strategies. Priming presents stimuli to a user and expects other stimuli in a different modality whereas cueing expects stimuli in the same modality.

Multimodal Interaction Systems

Although haptics has become an interesting field, graphical and auditory rendering are more common in current interaction systems. Even when haptic rendering enhances dramatically the quality of an interaction, it presents new challenges. Conveying information about a virtual object as stimuli is computationally demanding and the physical modelling of the interacting objects is complex. Nevertheless, a multimodal environment rendering both visual and haptics information improves the user experience into a more realistic immersion in interactive systems.

Physical modelling involves the representation of virtual object's physical properties, i.e. mass, stiffness, dynamic behaviour, and interactive forces among many others. Representing rigid objects is commonly used although deformable shapes have become of more interest in current research where nonlinearity is present in many physical properties. Refer to Terzopoulos et al [1987], Mahvash and Hayward [2004], and Faraci [2005] among many others.

In addition, surface and texture modelling together with friction representation is essential for some application as presented by Kim [2004].

The next section describes a simple architecture of a virtual reality system and a more detailed explanation of the most common force-feedback devices used in research for surgical simulations.

Architecture

Salisbury et al [2004] describe a basic architecture for a virtual reality application featuring visual, audio, and haptic feedback. Figure 1 depicts the main architecture components. The simulation engine is responsible for the computation of the simulation behaviour over time. Audio-visual and haptic rendering compute and convey graphic, sound and force responses to the user. Transducers help to convert the rendering output into a form the user can perceive. Displays, speakers and force-feedback devices are commonly used.

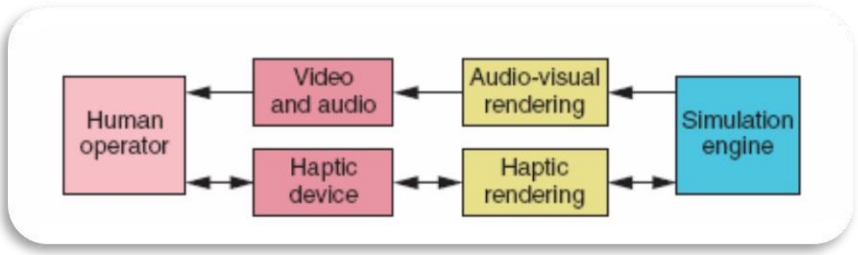


Figure 1 - Virtual Reality Application Architecture

Clearly from the diagram, there is a key difference in the information flow. Video and audio information flows in a unidirectional way towards the user whereas haptic information flows in a bidirectional way. While interacting with a haptic interface, the user either perceives a force from the simulation engine or sends its position due to a movement in the interface. This modality feature becomes an essential mechanism for the interaction in real time.

Haptic rendering algorithms are responsible to compute the interaction forces between the virtual objects and the virtual representation of the haptic interface in the virtual environment. This virtual representation of the haptic interface used by the user is called *avatar*.

Salisbury et al [2004] also identify 3 main components comprising a haptic rendering algorithm as shown in Figure 2, i.e. collision detection algorithms, force response algorithms and control algorithms.

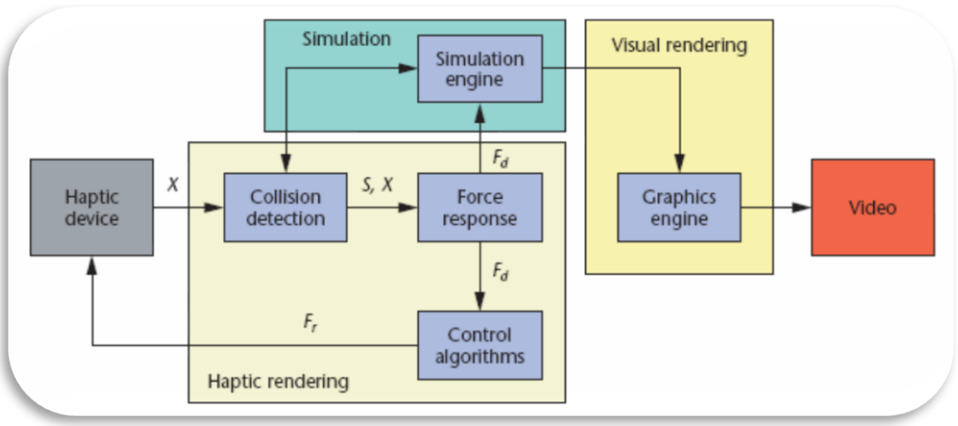


Figure 2 - Haptic Rendering Algorithm Components

Collision detection algorithms detect when, where and in what conditions a collision occurred between the avatar and objects. An avatar in the form of a tool with a tip provides a helpful simplification because the rendering is done mainly in the avatar's tooltip position *X* and the object in contact. Force response algorithms compute the interaction forces due to the collision as it would be normally done in the physical world. The positions *X* of the avatar's tooltip and the object as well as the collision state *S* between them are necessary to compute force and torque vectors F_d. These force and torque vectors are sent to the simulation engine where the mechanical behaviour is computed in a time-discrete approximation. Additionally, the interaction forces are processed by control algorithms which minimize the error between ideal and applicable forces F_r due to haptic device limitations.

The discrete-time nature of this approach is essential to approximate the dynamic behaviour of the environment. However, the only way to detect that both objects have collide is by means of the penetration of the avatar's tooltip into the object owing to the discrete-time nature. Several issues arise from this perspective, especially when dealing with thin objects. Some algorithms extensions or variations have been proposed extensively and will be discussed later in this document.

Haptic rendering algorithms are carried out in this sequence in a loop at each time step. Usually, a 1KHz servo rate is common for some applications. Although higher frequencies can provide texture sensations a trade-off is always considered due to constrained computational resources.

Collision-Detection algorithms

Collision detection has been studied for many years in computer graphics, especially for rigid objects in animation. However, collision detection for deformable objects is an important area of research. Cloth simulation, surgical simulation and animation are the main applications.

Teschner et al [2005] analyse various aspects that should be addressed for collision detection with deformable objects in contrast with rigid objects. In addition to contact points, self-collisions should be considered (usually neglected for rigid objects). Efficient algorithms use spatial data structures which are built in a pre-processing stage. These structures performed well for rigid objects, although they need to be frequently updated for deformable objects. Additionally, it is not sufficient to detect the interference of objects for deformable bodies, more information such as penetration depth should be provided as well. Finally, performance is of paramount importance for interactive applications of deformable objects.

Teschner et al [2005] describe different approaches for deformable objects, including *bounding volume hierarchies* (BVHs), *stochastic methods*, *distance fields*, *spatial subdivision* and *image-space approaches*.

Bounding Volume Hierarchies (BVHs)

In a pre-processing step, a bounding volume BV tree is recursively built until a leaf criterion is met based on object primitives. Primitives are the graphical entities that comprise the object. BVH works well with triangles and tetrahedra. The different types of BV are shown in Figure 3.

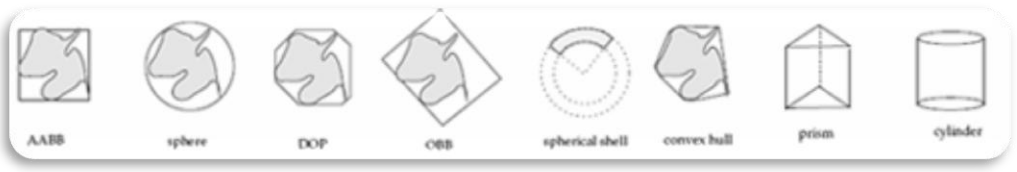


Figure 3 - Bounding Volume Types

During each time step, a top-down hierarchy traversal is carried out recursively to test overlaps between pairs of tree nodes. Furthermore, for deformable objects, the hierarchy structure needs to be updated/rebuilt, thus, efficient algorithms need to be considered. Self-collisions are easily implemented using BVH as the overlap tests can be done between nodes within the same object tree. The selection of the BV type, the arity of the tree, the creation of the hierarchy and either rebuilding or updating are some of the design features of BVH. For deformable objects, AABB type is usually selected because it can be refitted efficiently, 4-arity and 8-arity trees introduce a better overall performance and refitting is preferred over rebuilding.

Stochastic methods

Stochastic approaches give higher priority to the performance of collision detection algorithms rather than its accuracy, thus no exact simulation is possible. Two main methods based on probabilistic principles are described in Teschner et al [2005]. The *average-case approach* estimates the possibility of a collision with respect to a quality criterion based on the number of intersected polygons, whereas the *randomly selected primitives* approach guesses the potential intersecting regions.

Distance fields methods

Distance fields methods represent a highly robust collision detection alternative for non-interactive applications. Basically, space is divided by specifying the minimum distance to a closed surface for all the points in the field. *Uniform 3D grids*, *octrees*, and *BSP-trees* are used to represent distance fields. In deformable bodies, by comparing the vertices of one object against the distance field of the other object and vice versa, a collision is detected if the distance field mapping is less than the zero level set or sometimes less than an offset to avoid artefacts as described by Teschner et al [2005].

Spatial subdivision methods

For deformable object collision detection, Teschner et al [2005] analyse *spatial hashing* with uniform grids and tetrahedral meshes. This algorithm maps vertices into small grid cells in a hash table, providing efficiency in memory and flexibility. Then, information of the grids that were touched by tetrahedra is mapped. Finally, vertices and tetrahedra are checked within a hash table entry for intersections.

Image-space approaches

These methods process projections of objects to accelerate collision queries. Although especially targeted for environments with deforming objects due to the lack of a pre-processing stage, these approaches present drawbacks for physical-based simulations as collision response information is limited.

Force-response algorithms

As described previously in this document, tactile and kinaesthetic information are used in perception and become the basis of computing interaction forces. Salisbury et al [2004] distinguishes two different kinds of forces: forces due to *object geometry* and forces due to *surface properties*.

Several approaches exist to compute the interaction forces due to object geometry which includes *vector field methods*, *god-object and proxy algorithms*, *force shading* and some variations and extensions of these ones.

Vector field methods

Vector field methods are based on a one-to-one mapping between force and position. There is no record of previous avatar positions and hence difficult to handle thin or small objects where undetected collision may occur.

God-object and proxy algorithms

God-object and proxy algorithms are based on a *god/proxy* object connected by a spring to the avatar. In free space, the avatar and the god/proxy object are collocated, thus not returning any force. The location of god/proxy object is based on a set of active constrains as described by Salisbury et al [2005].

Force shading

Force shading is used to perceive smooth changing forces by interpolating normals. It is equivalent to Phong shading in graphic rendering which gets smoother surfaces by interpolation.

Various approaches are also available for computing the interaction forces due to surface properties which includes *Karnopp model*, *Bristle model*, *reset integrator model*, *texture mapping*, and *friction*.

Control algorithms

As described previously, control algorithms are useful to minimize the error between ideal forces and applicable forces due to hardware limitations. Basically, these limitations come from *device mechanics* issues and from the *virtual reality nature* of a simulation.

Device mechanics issues are caused mainly because of a limited magnitude of the forces than can be applied in a haptic device. Also, forces are not properly done in all directions. In addition to these issues, most haptic devices include friction, inertia and backlash which diminish substantially the ideal behaviour of the device in both free space and while in contact with an object.

Issues due to the virtual reality nature mainly consist of the time discrete feature in haptic devices whereas the user interacting with the haptic interface acts in a continuous time. Additionally, quantization errors arise owing to the finite resolution of haptic device sensors.

Finally, energy leaks affect stability, and some techniques have been designed to overcome unstable behaviour. *Virtual damping* limits the energy flow from the environment to the user. *Virtual coupling* uses stiffness and damping to create a link between the haptic device and the avatar limiting the maximum impedance that can be done by the device. *Multi-threading* is used as well with the aim of increasing servo rates by decoupling force response algorithms from other slower algorithms.

Unrealistic scenarios are present as a consequence of not overcoming the issues described above. In applications such as surface rendering, Choi and Tan [2004] describe *perceived instability* as all those unrealistic scenarios such as *buzzing*, *aliveness* and *ridge instability*. Buzzing refers to high-frequency vibrations a user feels when touching a textured surface. Aliveness regards to pulsating force changes even when the user has no movement when touching the object's surface. Finally, ridge instability is related to the perception of force directions. When the avatar's tooltip is in contact with a ridge in the surface it is actively pushed to the valleys.

Hardware Interfaces

To date, various haptic interfaces have been extensively used in research. It mainly refers to the physical connection between user and device through which energy is exchanged. Stylus interfaces and hand interfaces are the most commonly used.

Haptic interfaces may be classified by their grounding locations or more precisely by their mechanical behaviour or by the number of *Degrees of Freedom* (DOF) they can handle.

Regarding to the mechanical behaviour, an interface can be either of *impedance* or *admittance* type. Impedance devices read position and then send force whereas admittance devices read force and send position. Devices classified by the DOF of motion, i.e. the total number of dimensions they can handle, can be either passive or actuated, or either sensed or not sensed.

SensAble®'s Phantom Omni Haptic Device

SensAble®'s Phantom haptic device, shown in Figure 4, has been widely used in research for many years, especially in surgical simulators. It provides a 6-DOF positional sensing using a stylus.

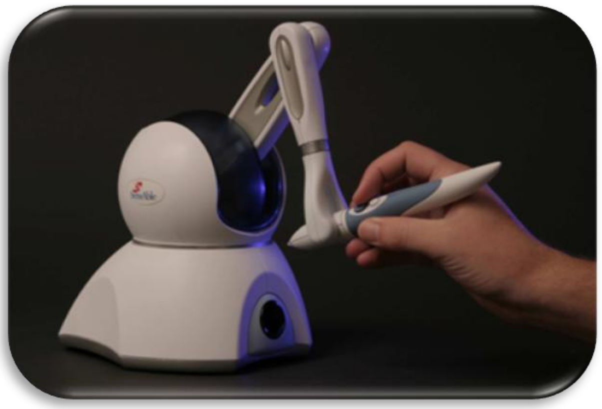


Figure 4 - SensAble's Phantom Omni Haptic Device

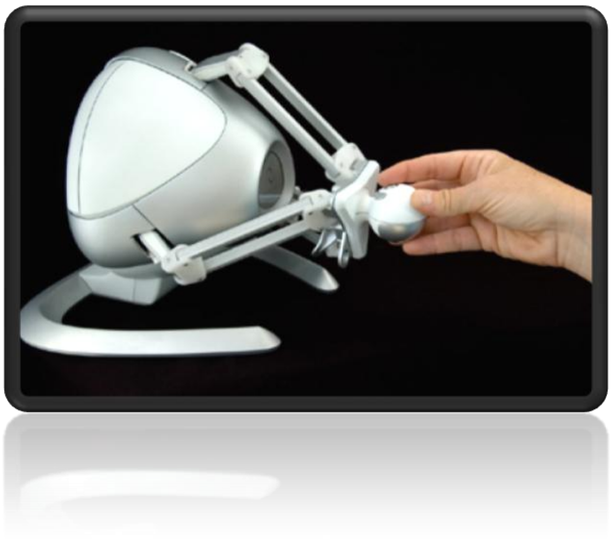
SensAble® provides a OpenHaptics Toolkit based on OpenGL API and X3D which will be discussed later in this document. The OpenHaptics toolkit is comprised of Phantom Device drivers (PDD), Haptics Device API (HDAPI) and the Haptic Library API (HLAPI).

NOVINT®'s Falcon Haptic Device

Recently, NOVINT® releases one of the most exciting consumer-based haptic devices, i.e., NOVINT®'s Falcon, as shown in Figure 5. Nowadays, it can be purchased for around £100. The advantage of price has leveraged both research- and consumer-oriented applications.

Falcon haptic device consists of a 3D cursor with detachable grips. It is a 3-DOF for both input and output, although more degrees of freedom are possible through enabled grips. Currents actuate motors at 1KHz.

Figure 5 - NOVINT®'s Falcon Haptic Device



NOVINT® provides an SDK API which can be used using C++. The API is divided in three layers: DHDLC (low-level drivers), HDAL (mid-level device communication) and Falcon API (high-level programming toolset).

Nonlinear Haptic Response of Deformable Objects

Thus far, a broad outline has been described for haptics above. Notwithstanding the advantages and challenges imposed by haptics, the introduction of nonlinearity behaviour and the modelling of deforming objects, such as soft tissue, increase the complexity of the application design.

As above, haptic rendering algorithms convey physical properties of a virtual object by computing interaction forces when in contact. Regarding a deformable body, Mahvash and Hayward [2004] describe that the computed force response due to deformation is a function of *deflection* only, which refers to the existing displacement of the initial point of contact of a deformable body when touched by the avatar's tooltip or a collision with another virtual object occurs. This principle holds without depending on the deformable body properties even if it is anisotropic, not homogeneous, nonlinear elastic or large deformation occurs on it.

In a virtual environment, the location of objects, their geometry, texture and friction are some basic properties that need to be modelled and considered by haptic rendering algorithms. Both *rigid* and *deformable* bodies present these properties. For deformable bodies more properties are involved though. Human organs properties, for example, consist of the type of material, support and the internal structure among others.

Furthermore, some properties differ depending on soft tissue conditions, i.e., either if it is attached to a bone or other organs, whether it is a in vivo sample or not, or even if an organ is filled or empty as in the case of the gallbladder. These properties are difficult to sample as they are different in one patient over the time and from patient to patient. To date, researchers have placed great emphasis on this area to avoid sampling in vivo and be able to model it physically as closer as possible.

In this sense, *high-fidelity* is mandatory for some application, especially simulation engines targeting deformation of soft tissue. High-fidelity implies that the virtual environment is represented with the fewest defects possible. Mahvash and Hayward [2004] list some simulation properties that are shared with the physical world. Firstly, the computing forces should reproduce the actual interaction. Secondly, the discrete nature of the environment makes difficult to obtain forces that are continuously related to displacement. Also, the principle of conservation of energy should comply as it is done in the physical world. Finally, the update rate of the simulation should be high enough to have a closer approximation to the real world.

Commonly, the modelling of realistic computed force responses is carried out by solving the *continuum equations of deformation*. Finite element methods and its variants are commonly used. In the following sections a more detailed explanation of these and other approaches can be found for deformation models.

Deformable Object Modelling

Deformable Models for Soft Tissue Simulation

The deformation modelling of objects in a simulated interactive environment is an active research area. An inherent *mechanical behaviour* is present in deformable objects and understanding the motion when forces are applied on them involves the necessity of a *physics-based modelling* in the simulation. The *dynamic behaviour* of these models is approximated by formulating and solving the continuum equations of motion, which are obtained from *Newtonian mechanics*.

Demetri Terzopoulos is one of the pioneers in integrating dynamics of deformable active models into a simulation. In computer graphics, *active models* are based on principles of mathematical physics. Terzopoulos et al [1987] used the *theory of elasticity* to model physical properties such as tension, rigidity, mass and damping. In a time discrete basis, the partial differential equations are numerically solved and hence be able to govern the evolving shape and its motion through space.

Surgical simulation for training in virtual environments is a common application found in research. It is computationally demanding because of the *complex nonlinear, viscoelastic, anisotropic and nonhomogeneous behaviour* present in human tissue. Furthermore, real-time is necessary to be able to simulate realistic interactive speeds for vision and haptics. Therefore, neither accuracy nor speed can be compromised and trade-offs are frequently considered.

Some methods rely on offline pre-computation of large number of possible exerting forces looking them up in real time. Although these methods are effective, they are not suitable for representing nonlinearity features. Nonetheless, there are approaches that adapt them to include nonlinearity as proposed by Mahvash and Hayward [2004].

Online methods using real time computation are most frequently used. These approaches use numerical integration methods through time in the form of small time steps during the simulation. In the following section a more detailed description of these methods can be found.

Deformable Models

The most frequently used deformation models are classified in three basic groups accordingly to Meier et al [2005]: heuristic methods, continuum-mechanics-based methods, and hybrid methods. Multi-resolution approaches are considered as a fourth classification group by Faraci [2005].

Heuristic methods

Mass-Spring Systems

This approach simulates a solid object through its surface. Using a mesh, a set of nodes with a given mass is interconnected through edges that behave as springs with given stiffness and damping.

MSS is not as demanding as continuum-mechanics-based methods, computationally speaking. Newtonian laws of motion are used to simulate the dynamic behaviour. Hence, a system of differential equations is solved for every node by discretisation of time in time steps. *Euler* and *Runge-Kutta* solvers are frequently used for this purpose.

Even when it is computationally attractive and a topology modification can be easily done, the disadvantages outweigh the advantages as analysed by Faraci [2005]. The lack of volumetric information does not allow for simulating cutting, which is a common task in surgery. The *central difference scheme*, as the integration method, presents a major drawback when handling the local mesh structure, that is, the displacement of one node only propagates to the closest connected nodes. Consequently, it does not allow a rapid propagation of deformations as at each time step. Also, MSS does not allow accurate modelling of material properties which are heuristically determined and oscillations may occur.

Nonetheless, MSS are typically used in medical and engineering simulations where a perfect simulation is not a constraint. As an example, Nedel and Thalmann [1998] used MSS to represent muscles at two different levels: action lines and muscle shape. More recently, Georgii and Westermann [2005] exploit features of *Graphics Processing Units* (GPUs) using two different data structures approaches to synthesize the interaction forces: *point-centric* (PCA) gathers information from adjacent nodes whereas *edge-centric* (ECA) uses scattering to update the position of mass points.

Linked volumes

Volumetric extension to MSS which represents the volume of a deformable object with evenly spaced cubic elements that are commonly interconnected using springs and dampers. It is more computationally demanding as MSSs due to the increased number of nodes and interconnections, therefore slowing down the propagation of deformations.

Chain mail algorithm

As linked volumes, this algorithm discretises volume, although the interconnection of the cubic elements is represented as links of a chain. In this sense, a link can moved feely without affecting any of its neighbours in a limited range and at the same time it only affects displacement of adjacent links. In this algorithm, force responses are assumed to be proportional to the penetration depth, thus non-homogeneous tissues are not properly modelled, as described by Meier et al [2005].

Mass-Tensor models

This method is another volumetric extension of MSS which represents volume as tetrahedra. Mass-tensor models provide a simplified linear relationship to describe the deformation behaviour in tetrahedra. Consequently, the physical realism of a simulation is limited to small deformations as analysed by Meier [2005]. Non-linearity variants have been proposed with the sacrifice of computational resources.

Continuum-mechanics-based methods

Boundary Element Method

The approximation of solving the differential equations of motions is applied in a specific domain. The boundary of an object corresponds to its surface, which is discretised in a finite number of elements as in a mesh. Hence, there is no need to involve internal points as part of the solution. Accordingly to Wu et al [2001], BEM is still dense even though it is one dimension less than FEM. Also, the internal behaviour of soft tissue cannot be modelled.

BEM evolved from integral equation methods known as *Boundary Integral Equation Methods* (BIEMs). From the two different approaches for formulating problems, the direct one is most widely and recently used in scientific applications. It is based on finding *Green's function* solutions of partial differential equations. Therefore, if the Green's function along with the boundary conditions of a well-defined mesh are known, then the solution of such boundary is also known as an integral equation that can be numerically computed, as described by Kythe [1995].

With some restrictions and assumptions on the deformable body, that is isotropic, homogeneous and linear elastic behaviour, Monserrat et al [2001] use efficiently BEM and ongoing research is focusing on the application of viscoelasticity and nonhomogeneous properties as found in soft tissue. Viscosity is represented by nonlinear strain in the velocity field as stated by Wu et al [2001].

Finite Difference Method

In FDM, a differential operator is approximated by algebraic difference operators. This approach provides accuracy and efficiency when the geometry of the object is regular.

Finite Fast Elements

FFE simplifies a volumetric mesh by compressing the system to surface points. FFE is a fast approach for real-time applications using linear models. Cutting cannot be modelled by FFE because it requires offline computations, thus topological modifications are not considered.

Bro-Nielsen [1996] overcomes the problems of a surface model by using volumetric data and FFE for surgery simulation. As a result, a linear matrix system of equations is computed based on Hooke's law, which models linear elasticity of deformable objects.

Finite Element Method

FEM are often preferred by researchers in solid mechanics and structural engineering as it allows to discretise irregular geometries. Conventional FEM is computationally demanding in processing and in memory. Active research is constantly done to reduce the computation overhead for real-time purposes in simulation.

Discretisation transforms the partial differential equations of motion into a system of linked ordinary differential equations, as analysed by Terzopoulos [1987].

Linear models were initially introduced because they are fast and allow off-line pre-computation. Linearity assumes both linear elasticity and linear strain. The linear system of equations is represented as follows:

$$M\ddot{u} + D\dot{u} + Ku = f$$

where mass matrix *M*, damping matrix *D* and stiffness matrix *K* are all constant and usually sparse. The displacement vector is represented by *u* and force by *f*.

Wu et al [2001] describe nonlinear FEM in two nonlinear proportionalities, i.e., strain is nonlinear in velocity and displacement whereas stress is nonlinear in strain. The nonlinear system of equations is represented similarly as the linear version, although stiffness is no longer lineal:

$$M\ddot{u} + D\dot{u} + R(u) = 0$$

Essentially, this system of second-order ordinary differential equations is computationally demanding and storage-consuming. However, due to the accuracy of the model representation, FEM is frequently used for modelling soft tissue.

There are some well-known techniques that help to speed up the computation time emphasising in real-time feasibility. *Mass lumping* and *time integration* are frequently used in FEM with a single processor.

Mass lumping produces a diagonal matrix. One of the simplest methods is column/row summation. The total mass of the object is preserved and it allows local mesh refinement in real-time. Even better, as shown in Zienkiewicz and Taylor [2000] dissipation is added to the stiffness matrix whereby oscillations are cancelled out.

There are two types of time integration techniques: implicit and explicit. On the one hand, implicit methods are commonly found in animation using large time steps. For real-time applications it is not effectively used because of a poor convergence time, though. On the other hand, explicit schemes are faster, although they require smaller time steps to avoid instability. Zienkiewicz and Taylor [2000] describe a critical time step to guarantee stability, which is directly proportional to the total number of elements and inversely proportional to the square root of the stiffness.

Recently, Dehghan et al [2007] use FEM for fine-tuning purposes while modelling needle-tissue interaction using ultrasound radio-frequency for motion estimation using tetrahedral elements.

Bargteil et al [2007] were even further using FEM for more sophisticated models. They incorporate both plastic and elastic deformations for simulating viscoplatic flow. These properties are commonly found in everyday life such as chewing gum, toothpaste, and shampoo.

Multi-resolution approaches

In a multi-resolution approach, different levels of detail are applied to the mesh. High resolution is presented in areas of deformation whereas a lower resolution is presented far from the contact area. At the same time, the physical behaviour improves by adding more elements in the contact area. Faraci [2005] proposes a nonlinear multi-resolution scheme by defining two levels of resolution for tetrahedral meshes: *coarse* for those areas not in contact and *refined* for deformable areas.

Furthermore, selection of active tetrahedral, online remeshing and refinement throughout the simulation are common tasks done in this kind of approaches, as proposed by Paloc, Faraci and Bello [2006]. A more detailed description can be found in the following section.

Time discretisation effects

Finally, it is worthy to comment that the approximation of a physical problem merely affect the simulation as the result of the time discretisation in numerical integration.

Passivity is a physical property that defines that some objects are incapable of generating energy by themselves. Passivity objects in a virtual environment unavoidably add a measure of activity owing to the discrete time nature of the simulation.

Mahvash and Hayward [2004] detect buzzing, aliasing of force signals, and low-frequency harmonics that are not filtered out by the haptic hardware as effects of time discretisation. This problem can be solved by increasing the rate of the simulation. However, they suggest a multirate approach instead.

Remeshing Deforming Objects

As it was described in earlier sections, triangular mesh representations are used to model the surface of virtual objects. Triangular meshes are efficient structures for graphical rendering. However, physical modelling requires a more appropriate representation and tetrahedral mesh structures are used instead.

The main advantages of using tetrahedra include the following. Cartesian, rectilinear and curvilinear meshes can be represented using tetrahedral meshes. Tetrahedral meshes produce a more detailed representation using fewer elements. It provides a surface triangular mesh representation. In MSS, they are suitable to preserve volume. Finally, tetrahedral meshes are extensively used in FEM.

Remeshing techniques are useful in multi-resolution approaches where local refinement is done in contact areas that are likely to be deformed. The deformation of the object then requires a higher resolution which involves online remeshing, either by inserting new mesh points or by removing them in applications such as cutting in surgical simulation.

In either case, the complexity becomes apparent when a good enough mesh quality is required for the numerical integration. Degradation may happen in the simulation if the elements in the mesh include small or large angles. Therefore, *Delaunay criterion* is used to guarantee the quality of the tetrahedral mesh. It is sometimes called *empty sphere property*. A Delaunay triangulation maximises the minimum of the angles formed by the edges between the elements.

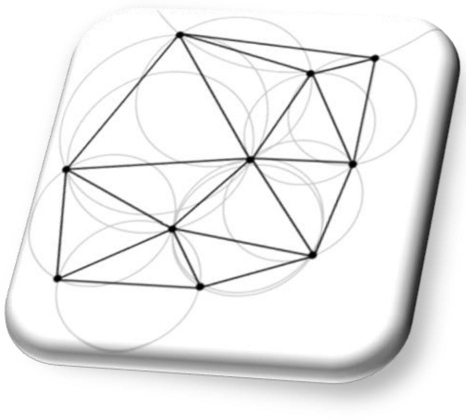


Figure 6 - Delaunay Circumspheres

This property states that any node must not be contained within the circumsphere of any tetrahedra in the mesh. A circumsphere passes through all four vertices of the tetrahedron. A common technique to restore the Delaunay property is by *flipping algorithms*, either using edges or faces.

Once a local area has been refined, it is unlikely to require high-resolution at a later stage in the simulation. The simplification procedure representing the reverse operation is called *local mesh decimation*. Paloc, Faraci and Bello [2006] used these algorithms for online remeshing of soft tissue.

Finally, Wu et al [2001] use *dynamic progressive meshes* for nonlinear deformable bodies, which consist of offline pre-processing and online refinement. In the offline stage, a good quality and detailed meshed is provided and then coarsened. The process is stored in a hierarchical tree which is useful in the online stage. Error estimation determines where local mesh refinement is necessary. Wu et al [2001] identify four error estimators which are based on stress concentration, displacement field, substantial internal stress variation (optimised posterior), and stress gradient.

Open Source Software Frameworks

SOFA: Physical Simulation Framework

Introduction

SOFA's prime objective is focused in *medical simulation research*, providing a common *software framework* for the community. Figure 7 shows a primal target application for medical simulation for laparoscopy.

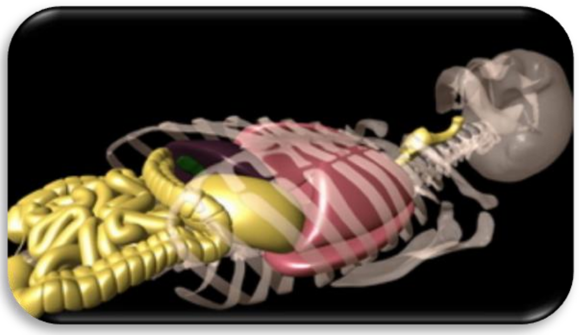


Figure 7 - SOFA Laparoscopic Surgery Simulator

The key difference of open source C++ library among other open source solutions is the *modularity* and the *flexibility* of its architecture design. Furthermore, independent developed algorithms can be incorporated within the same simulation engine minimising integration time. Additionally, it allows the modification of simulation parameters such as deformable behaviour, surface data representation, integration solver, constraints, collision algorithm, and so on.

Design

Scene-graphs, commonly used in graphics, are *directed acyclic graphs* introduced in SOFA. They permit the creation of complex models by organising and processing the elements of the simulation. Each component is attached to a node in a tree structure. Figure 8 shows a scene-graph for a pendulum with a deformable string and a rigid object. Nodes are represented as grey hexagons and components as coloured squares (colour indicates the component type). Thick arrows represent tree hierarchy, thin arrows represent attached components to the nodes, and dotted arrows are pointers between components. Faure [2007] describes this example in a software perspective.

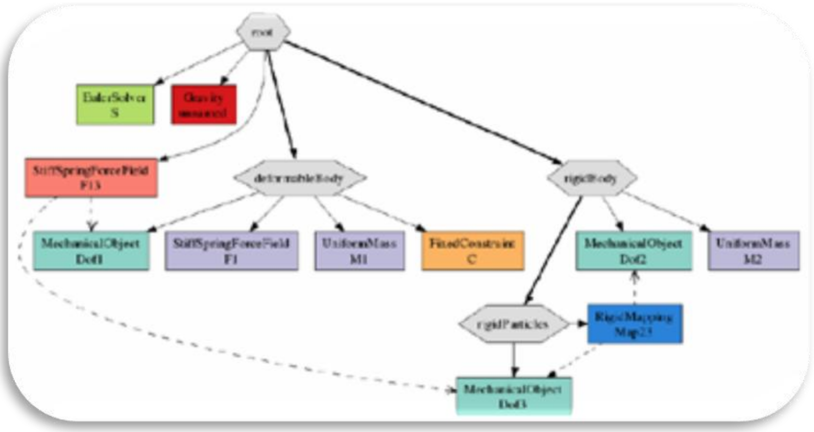
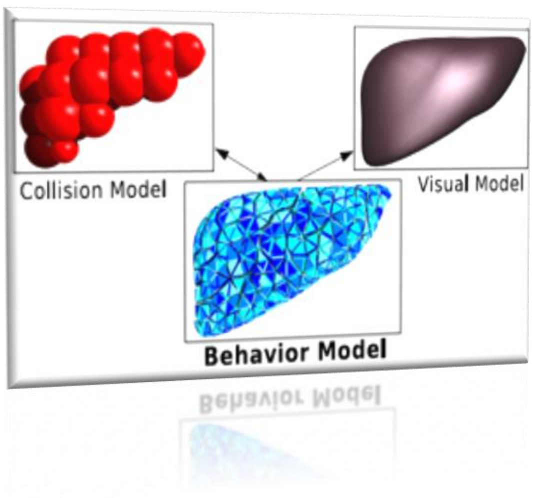


Figure 8 - SOFA Pendulum Scene-Graph

Two levels of modularity are present in SOFA: multi-model representation and fine granularity level for physical properties, as described by Allard et al [2007].

SOFA architecture supports a *multi-model representation*, i.e., simulation components can have several representations connected together through *mapping* whereby allow us to use different geometric models for a given body. Figure 9 shows an example of different representations of a liver (mappings are depicted as arrows). Multi-model representation is the basis of a *high-level modularity* scheme. As an advantage, it is possible to have models of different nature (rigid, deformable, and fluid objects) interacting together.

Figure 9 - SOFA Multi-Model Representation



SOFA also introduces a *finer level of granularity* for physical models, i.e., the *Behaviour Model*. Thus, physical models are decomposed into a set of basic primitives: `DoF` (positions, velocities and accelerations), `Mass` (scalar, vector or matrix), `Force Field` (springs, linear and co-rotational FEM, Mass-Tensor, and Smoothed Particle Hydrodynamics), and *ODE* (Ordinary Differential Equation) `Solver` (Euler, Runge-Kutta 4, and implicit conjugate-gradient based Euler). Switching among available components allows its comparison and further analysis.

During the loop simulation, the data structure is processed using *actions*, which recursively traverse the tree hierarchy calling appropriate *virtual method*s. Traversals can be done in two ways: *top-down traversal* (TDT) and/or *bottom-up traversal* (BUT). `InitAction`, `AnimateAction`, `VisualDrawAction` and `MechanicalComputeForceAction` are some examples among others. Actions belong to namespace `simulation::tree`.

Implementation

SOFA framework is distributed under the terms of *GNU Lesser General Public License* (LGPL) which permits its use solely in proprietary programs. Currently developed by four research teams in two institutes: INRIA (Alcove, Asclepios, Evasion) and CIMIT (SimGroup, MIT/Harvard/MGH). The latest release (as per the time of writing this document) is *version 1.0 beta 2*.

Simulation parameters can be stored in an XML file which is parsed using *libxml2*, which is an XML C parser and toolkit developed for the Gnome project. The parsing process is handled by the `sofasimulation` component under the `sofa::simulation::tree::xml` namespace.

SOFA supports two default user interfaces: *Qt* by Trolltech, and *FLTK* (Fast Light Toolkit). They are implemented in SOFA as `sofaguiqt` (`sofa::gui::qt` namespace) and `sofaguifltk` (`sofa::gui::fltk` namespace), respectively. Both provide a portable windows-handling mechanism to interact graphically with SOFA. Event handling is done through *callback procedures* and *widgets* are used to design the window layout.

In addition, *OpenGL* and *Ogre* graphic renderers are supported. Both rendering engines are widely used in a myriad of application.

Independently, Gmsh software has been used to visualise available human tissue data in '.msh' file format. 3D point coordinates are set as nodes and then referenced to create tetrahedra elements. This representation is loaded in the topology component (`sofacomponent`) using the `MeshTopology` class.

Finally, *NewMAT* C++ library is used by SOFA. NewMAT provides the manipulation of a variety of types of matrices using standard operations. It is part of `sofadefaulttype` component library.

SOFA Future

As described previously, SOFA presents an efficient solution for medical research simulators. Nonetheless, SOFA presents few weaknesses that need to be solved. Documentation is incomplete, the main loop is not customisable, and constraint-based methods are difficult to implement.

Accordingly to Faure [2007], future work within SOFA development team involves coping with the above weaknesses and implementing features such as friction, haptics, and faster collision detection algorithms, among others. For instance, SOFA is integrating haptics to their framework using SensAble® devices and even at ISBM'08 they mentioned this dissertation project as an ongoing implementation to handle haptics using H3D.

The roadmap also includes parallelisation using GPUs and clusters and an easier debugging process. Early attempts using spring-based models have used GPUs in SOFA accordingly to Faure et al. [2007]. They transfer part of the simulation to the GPUs using NVIDIA CUDA library.

H3D: Haptic Rendering Framework

Introduction

H3D API is an open source, cross-platform scene-graph API. It is developed by SenseGraphics® and provides a development framework for *multi-modal applications*, called sometimes *immersive systems*.

In research and in industry, H3D has been recognised as an ideal development platform. It leverages on a diverse range of haptic devices including products from SensAble®, Novint® and ForceDimmension®.

Design

H3D has been designed using many industry standards which include X3D (Extensible 3D file format) for the scene-graphing. X3D replaces the now outdated VRML standard. XML is included as well along with Xerces for X3D file parsing. Additionally, OpenGL library is used for 3D graphics. Finally, H3D is based on *STL Standard Template Library* which supports rapid development by using a large collection of C++ templates.

H3DLoad program is used to load X3D files. Configuration setting can be customisable by using *H3DLoad Settings*. *H3D Viewer* displays X3D files with haptic features. It allows the configuration of the navigation paradigms: *WALK, FLY, NONE, ANY, EXAMINE*. Only the first three paradigms support collision-detection as per its specification.

X3D is composed by *Nodes* and *Fields* which represent a structural division. Fields are data containers that store and manipulate properties. Nodes are containers and managers of fields. Nodes are the building blocks for traditional scene-graph APIs and they become useful to group set of fields for large field networks.

Fields are seen as fundamental blocks in X3D and hence in H3D. It can be used as an event handling mechanism. A *field network* is a directed graph structure that connects fields by means of *routes*. Events are passed through the directed network where information may be stored.

Fields can have several functions: data storage, data dependency (triggering updates based on other fields) and functional behavior (computing its value depending on the fields routed to it by overriding the `update()` function). In either case, the values are updated using *lazy evaluation*, i.e., update becomes when someone ask for its value.

For haptic rendering, a Surface node with haptic properties must be added to the *Appearance* tag in X3D. Current version supports the following surfaces: `SmoothSurface`, `FrictionalSurface`, `MagneticSurface`, and `OpenHapticsSurface`. *Force effects* are supported as well and represent a way to map a haptic device position/orientation to a force/torque. Force effects include `ForceField`, `SpringEffect`, `MagneticGeometryEffect` and `ViscosityEffect`.

Implementation

H3D is distributed under the terms of ordinary *GNU General Public License* (GPL) which permits its use only in free programs. The latest release available is version 2.0 alpha. H3D API is entirely written in C++ STL. API can be used using either C++ or Python.

H3D is easily customized to work with a wide variety of virtual reality display systems. Also, haptic device settings and graphical render settings can be easily changed. *H3DLoad Settings* program load these settings by reading X3D/XML files for the display type, the haptic device, the stylus shape of the avatar and the starting viewpoint of the scene.

OpenGL is used for graphic rendering. This implementation uses *GLUT OpenGL Utility Toolkit* which is easier to integrate than native OpenGL. GLUT in not open source and for this reason other alternatives are considered instead, e.g. FLTK and freeglut. For haptic rendering, H3D API uses SenseGraphics' HAPI.

Recently, Vidholm [2008] developed a library toolkit called *WISH – Interactive Segmentation with Haptics*. It is publicly available as open source code. It uses H3D API for the haptic rendering.

FLTK: Fast Light Tool Kit GUI

Introduction

As previously described, FLTK provides a cross-platform library for window handling and event handling. It uses widgets as the building blocks of a graphical interface. This section briefly describes event handling, FLUID and GLUT compatibility.

Design

Events are mapped to an integer number depending on the action performed by the user. Events can be any of the following types: mouse, focus, keyboard, widget, clipboard, or drag and drop. The event is passed to the `Fl_Widget::handle()` virtual method. Additionally, FLTK stores the information of the most recent event in static storage and it can be accessed by inline functions of the form: `Fl::event_*()`.

FLTK supports a GUI designer called FLUID Fast Light User-Interface Designer. It allows to easily create GUIs by using a pre-defined set of graphic components. FLUID then creates a '.fl' file storing the layout of those components and thereafter generates C++ source code. It is a flexible approach for a fast creation of GUIs.

Finally, it is important to emphasise the GLUT Compatibility available. As described earlier in this document, GLUT is not open source and some alternative solutions have been created amid we can find FLTK. Therefore, a GLUT window becomes a child of a `Fl_Window` object. `glutInit()` function should not be called. Once `FL_Window` is shown `show()`, the GLUT code is enclosed by `window->begin()` and `window->end()` functions. Either `glutMainLoop()`, `Fl::run()` or looping `Fl::wait()` can be called to run the program.

Framework Design

Introduction

Objective

The aim of this project is to develop a framework for medical training in pre-operational tasks. This framework is based on some topics covered as a background in the previous chapter. Thereafter, a key element in the design is to allow users to interact with deformable objects in a simulation environment by means of the sense of touch. Furthermore, depending on the amount of forces exerted, virtual objects should deform accordingly based upon an underlying mathematical model.

The framework, called ***Haptics Open Framework (HoF)***, is targeted mainly at medical staff to whom pre-operational training in a simulation environment becomes an important tool to perform efficiently in a surgery room.

Additionally, the aim is to build a framework for quick prototyping of procedures which involve deformation and which are indeed common in medicine. Furthermore, it is essential to provide a framework for testing different models regarding to deformation and haptics.

It is of paramount importance to integrate open source libraries to build the framework. Moreover, such integration should allow the developer to both get the best of the flexibility and modularity of the libraries and extend the framework itself. Source code modifications of the used open source tools were considered as a minimum so the installation will not become overwhelming.

The graphical user interface becomes an important aspect of the project as this will be the interaction part for the user to handle the most important library settings used, especially for the mechanical behaviour, graphics and haptics rendering. The purpose of this component is to provide a straightforward method for interacting with virtual deformable objects while modifying its settings. Additionally, users are able to save and load project settings for a specific human organ model.

The last but not the least is to increase the number of frames per second in the simulation environment so the users will be able to interact in real time.

Considerations

In this chapter, several design features are described. Firstly, the hardware platform used for development and installation is described. Shortly thereafter, the software architecture is defined in a conceptual perspective and how the open source libraries have been integrated in an abstract level. Furthermore, reached design decisions regarding to the haptic rendering and deformation modelling are explained. Such decisions are supported by the tests carried out with analysis and results shown in the analysis and evaluation chapters.

Hardware Platform

Test environment

Haptics Open Framework is targeted to a PC environment with a dual core processor and a graphics card. Merely by using both SOFA and H3D libraries as well as the Novint®'s FALCON haptics device, HoF becomes a perfect solution against commercially available products and other more expensive haptic devices without the requirement to consider an expensive hardware platform.

HoF was developed, tested and installed in an Intel® Core 2 CPU 6700 running at 2.66GHz CPU speed with 2GB of RAM. Additionally, an NVIDIA® Quadro FX 3500 has been used which is the basic product in the high-end graphics solutions.

Software Architecture

A Conceptual Perspective

The purpose of this section is to describe in a conceptual perspective how the framework is composed to have an overall understanding of its software design. In an abstract level it consists of five different components as shown in Figure 10.

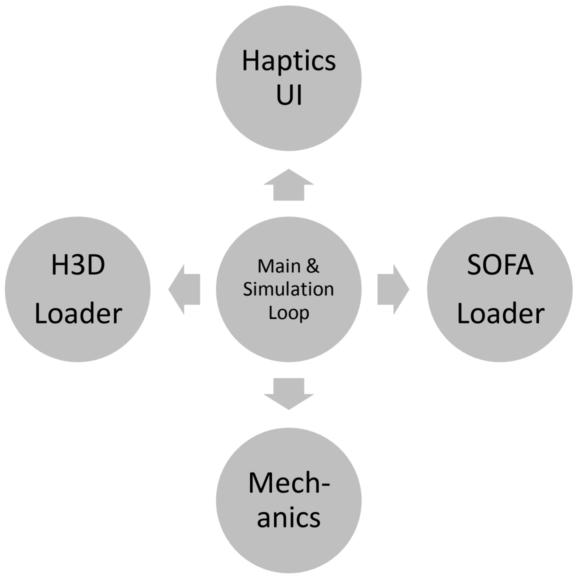


Figure 10 - Haptics Open Framework Design

The first module is further subdivided into two subcomponents. The “Main” sub module merely runs the graphic user interface loop for event handling and windows showing. It also declares three global variables that will be used throughout the simulation loop: the graphic user interface itself, the SOFA’s scene-graph root node and the mechanical object for retrieving the vertices position at each time step. The second subcomponent called “Simulation Loop” contains the deformation algorithm and animates the object’s mechanics at each time step. It is completely linked to the haptic rendering algorithm and detects when the avatar touches the virtual object so the exerted forces are sent to SOFA and the new deformed nodes’ position is updated into H3D geometry shortly thereafter. This is an important step for the deformation modelling as the user can see how the virtual object is deformed based on a mathematical model provided by SOFA mechanics.

The second module called “HapticsUI” provides the necessary widget objects to represent windows, input values, menu bars, sliders, browsers, tabs and many more. One of the key elements in this module is the `GLUTWindow` variable which is a customised H3D class that displays the X3D scene-graph. This window is embedded into the framework’s main window. There exist three more windows that are created for the mechanical behaviour, graphics rendering and haptics rendering respectively. Moreover, the main window contain buttons to handle the animation, shows frames per second rate and when the virtual object has been touched, relevant information is shown by using output value widgets. Among these values the user can see vectors such as touched positions, forces, and normals. Finally, it provides a way to load or save a new project in XML format by saving the whole scene-graph (haptic and deformation).

The third module called "SOFALoader" retrieves the mechanical behaviour settings from the HapticsUI widgets and recursively builds an XML document tree in memory which is sent to SOFA shortly thereafter. Such settings are used to create the SOFA scene-graph and initialise the SOFA components before the animation starts. Some of the components include solver, mass, force fields, constraints, context and topology among others. Finally once a project has been loaded into the framework, this module populates the mechanical behaviour settings into the UI widgets.

"H3DLoader" is another module and it is responsible to initialise H3D using the information held by HapticsUI in both the graphics rendering and haptics rendering windows. Information such as console settings, rendering mode, haptics device, stylus, viewpoints and X3D scene-graph are sent to H3D for the creation and initialisation of the Scene object. It also populates back these settings once a project has been loaded.

Finally, the last module called "Mechanics" is responsible to receive the touched points and forces applied to the object. It detects which vertex indices will be set in an interaction force field and which forces are applied to them so that SOFA computes the next vertices position as part of its numerical integrator.

Open Source Tools

Haptics Open Framework is a conglomeration of six open source libraries as depicted in Figure 11. Extensive analysis in each of the tools has been taken into account for their integration into the framework, especially for SOFA, H3D and FLTK. An overview of how these libraries have been combined together and have been used is described in this section.

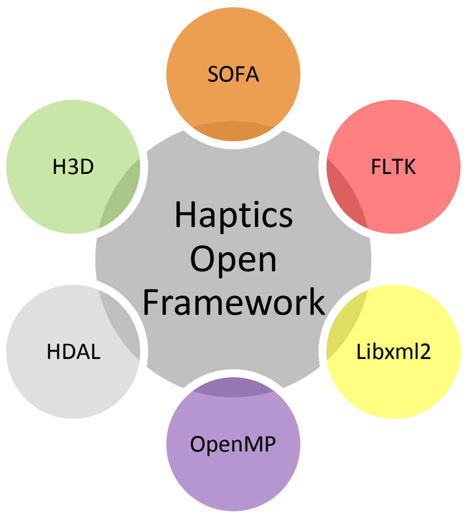


Figure 11 - Integrated Open Source Tools

Two UML sequence diagrams in Figure 12 and Figure 13 are used to explain the relationship among the libraries and the main calls and loops that are executed as an overall. The first diagram describes the interaction between the user and the framework itself before the simulation loop starts. The second UML diagram shows the basic steps that are executed once the simulation has been set up and the user starts to see the graphic rendering and manipulate the haptics device. The colours in Figure 11 for each of the source tools are intended to be used as a reference in the UML sequence diagrams in Figure 12 and Figure 13 so the reader can associate which module correspond each tool.

FLTK

FTLK is incorporated into the framework for two main reasons: as a replacement of Freeglut in H3D and to provide the graphical user interface.

On the one hand, even when Freeglut is open source, it does not provide GUI capabilities as FLTK does. `H3D::GLUTWindow` is a particular implementation of a glut window node in H3DAPI, therefore it represents the starting point to integrate FLTK. Basically, `H3D::GLUTWindow` becomes a child of `Fl_Window` in addition to `H3D::H3DWindowNode`. In order to guarantee that this window will be embedded into the main one, the windows are shown in order, that is, showing the main window first and then the embedded glut window.

On the other hand, FLTK handles the events and register the appropriate callbacks for the `H3D::GLUTWindow` especially for the keyboard and mouse events. FLTK's main loop starts by calling `Fl::run()` as shown in the diagram (message #2). FLUID plays an important role for the creation of the widgets (message #1).

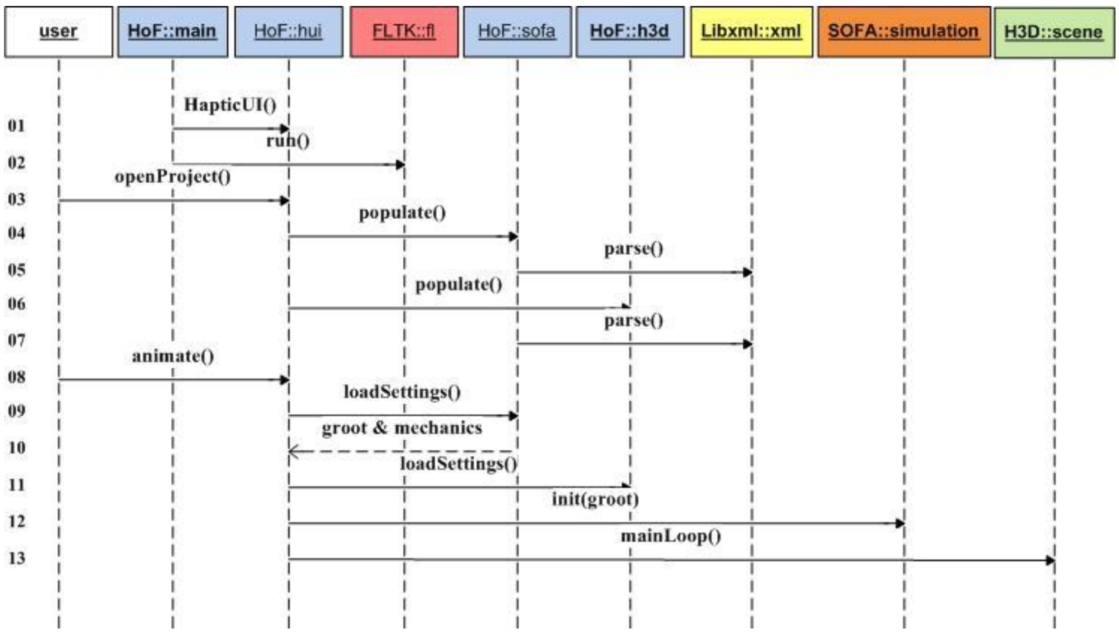


Figure 12 - UML Sequence Diagram 1

Libxml2

As shown in the UML diagram in Figure 12, XML is used to either create a project file or to parse an existing one (messages #5 and #7). In either case, an XML document tree is created in memory and parsed recursively for getting and setting XML properties in each node (messages #4 and #6). All framework settings are stored in an XML file. An example of an XML project definition file can be found in the appendix.

It is important to emphasize that SOFA scene-graph can be used without any modification under the XML node '*sg*' within the project as soon as the nodes are parsed by the GUI and loaded into the appropriate window.

HDAL

The Haptic Device Abstraction Layer is supported transparently by H3D for the Falcon device. By compiling H3D's HAPI project with the `HAVE_FALCONAPI` pre-processor definition, H3DAPI associates a stylus with the position of the haptic device tip. In every time step, device values such as force, torque, position, velocity, orientation and button status are queried from the device and updated (messages#14 and #15). HDL library provides a series of functions to setup these values and handle each haptics device.

H3D

H3D controls the main loop in the framework by calling the static function `mainLoop()` in H3D::Scene class (message #13). The Scene node is the topmost node that takes care of the rendering of the scene graph both haptically and graphically. The `Scene::idle()` function is registered as an idle callback and the main activities performed by it are shown in Figure 13 as an UML diagram.

Once graphical and haptics settings such as rendering mode, viewpoints, stylus, X3D scene-graph have been loaded into H3D (message #9), the scene is traversed in each execution cycle.

It is important to emphasize that the topology handled by H3D is merely a surface composed by a set of haptic triangles. The vertices of such triangles match the vertices found in the surface of the volumetric data used by SOFA.

Traversing the scene graph means rendering each node from the parent to all its children recursively. In the particular case of a deformable shape (message #16), there is a node called `DeformableShape` which inherits from a general X3D shape node. This node allows modifying the geometry of a shape when it has been touched with a haptics device. `HoF::Motion` is then created inheriting the properties of `H3D::H3DCoordinateDeformerNode` and permitting to handle the deformation modeling as required by overriding the `deformPoints()` function (message #17).

This is one of the key elements of the integration as `HoF::Motion::deformPoints()` function has enough information available to identify when, how and where the surface has been touched as well as the current deformed points which will be used to compute the new deformed points position.

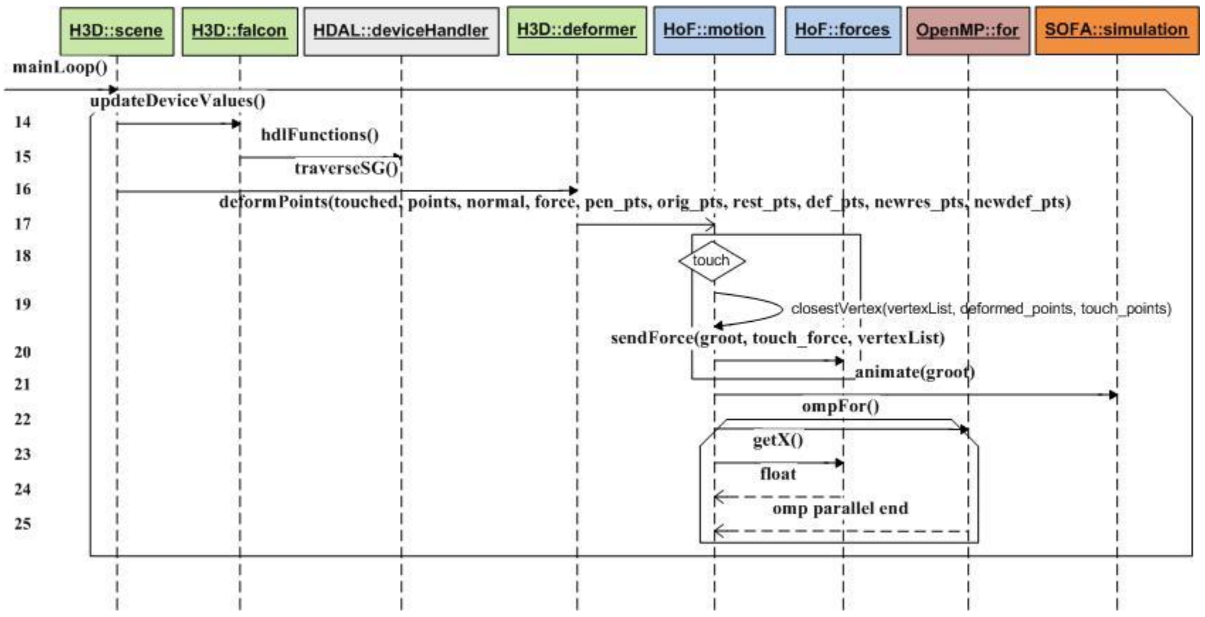


Figure 13 - UML Sequence Diagram 2

SOFA

Thereafter, once the simulation has been initialised (message #12), the detected forces from the Falcon device can be sent to SOFA's mechanical behaviour from the X3D Motion node (message #19). An interaction external force field component is used every time the virtual shape is touched (message #18); both the indices of the touched points and the forces are set as values of the force field.

Then based upon the scene-graph of SOFA, new vertex positions are computed as part of a numerical integration. The underlying mathematical simulation is animated each time step (message #21) defined in the scene-graph and called every time the Motion simulation loop executes. The mechanics of the object then is queried (message #23 and #24) to update the new points' position into H3D so each Coordinate point can be rendered accordingly (message #25).

It is important to emphasize that the topology of the mechanical behaviour is in the form of a mesh with a set of volumetric tetrahedra.

OpenMP

There are at least two different approaches OpenMP was used in the framework. The first one was targeted to the `Scene::idle()` function as a mean to improve the frames per second rate. This integration into H3D becomes difficult as most of the calls are recursive while traversing the X3D scene-graph. The main goal was to define section directives so the main tasks in the `idle()` function will be handled by independent threads and treated as functional parallelism.

On the other hand, even though the second approach was easier to design and implement, it was only supported in Visual Studio 2005 (vc8) and tested as a debug compilation. It merely defines a for loop directive for data parallelism (message #25). Therefore the mechanics of the deformed object will be updated in parallel into H3D.

All the libraries used by the framework have support for Visual Studio 2003 (vc7) but not all for 2005 and even when OpenMP supports starts in vc8, the Intel C++ compiler 10.1 can be used which has support for OpenMP and vc7. The latter approach was not tested in the framework.

Haptic Rendering

Haptic Rendering Algorithm

In HAPI, a haptic rendering algorithm determines the forces and torque depending on the haptics device position and all the shapes added to the scene. There exist internally implemented algorithms such as Ruspini renderer and external library algorithms such as OpenHaptics and Chai3D. Most of these algorithms are variants of a proxy-based model.

Proxy-based rendering defines the haptics device position as a virtual representation called proxy which always lays outside the geometry surface. Even when a haptics device has penetrated a surface, the proxy remains outside and the forces are computed as a spring.

In HAPI, the geometry of the proxy could be either a point or a sphere. On the one hand, by having a point, it becomes difficult to touch small or thin objects. On the other hand, by having a sphere it makes more difficult to constraint the proxy to the surface and more calculations are required.

Even when any haptic rendering algorithm can be chosen by the user in the Haptics Open Framework, Ruspini algorithm has been considered as part of its design. One of its main advantages is that it allows for variable proxy radius, thus a sphere is used as a proxy. It is a variant of the algorithm presented in Ruspini et al [1997]. In this paper, configuration-space obstacles are defined consisting of all the points within one proxy radius from the object surface. Then, constrained planes are computed depending on which element is in contact: none, one, two or three plane constraints for free space, a point, an edge or a vertex respectively. In the implementation chapter, a more detailed analysis and comparison of all the algorithms is described along with its implementation in H3D.

As part of the design of the framework, a haptic rendering algorithm which perfectly fits in dynamic environments is needed because of the nature of deformation shapes. The traditional proxy/god-object method proposed by Zilles et al [1995] is not appropriate at all as many fallthrough issues exists due to round-off errors in the newly computed god-object point position.

Ruspini algorithm has both advantages and disadvantages. On the one hand, it is an open source solution which is the key element of the framework. Also it is device independent, therefore the Falcon haptic device can be easily integrated as opposed to OpenHaptics rendering algorithm which is targeted only to Phantom devices. Additionally, the proxy is represented as a sphere with variable radius so fallthrough problems are minimised. Finally, haptic surfaces can be defined by the user to render properties of a geometric shape such as friction.

On the other hand, Ruspini algorithm is slightly slower than the others because of the constraints construction and variable sphere radius. In the evaluation chapter this was not perceived as a difference compared to the other algorithms. However, some fallthrough problems exist when moving the stylus over the surface when forces are applied. Even when the problem has been tackled by Ruspini algorithm, sometimes the proxy radius is not large enough to avoid gaps from polygons that share a common edge. This problem is caused by small numerical errors. Nonetheless, Ruspini algorithm has shown good performance during the evaluation and results. Therefore, it has been chosen as part of the framework design.

Collision Detection Algorithm

Collision detection is an important subcomponent in any haptic rendering algorithm. Its goal is to generate force and torque vectors to send them to the haptics device given a set of geometries and current device values.

Both SOFA and H3D handles collision detection and a comparison of them was considered for the purposes of this paper. Detailed information on how both libraries implement collision detection is found in the Analysis chapter.

The nature way of handling collisions in the framework is by using H3D because as the scene-graph is traversed collision objects are identified depending of the position of the avatar. However SOFA was analysed as an alternative solution for the framework and be able to determine which one performs better. Even though this activity was not developed, some topics were identified for its future integration.

Basically, SOFA scene-graph would be comprised of two nodes, one for the virtual object and one more for the avatar/stylus, each with a mechanical object. The avatar geometry will be defined as a sphere. The avatar's tip position and the force would be then sent from H3D to SOFA so external forces could be applied. The mechanical behaviour will be sent back to H3D accordingly. Finally a new haptic rendering algorithm should be created in H3D which will communicate with SOFA's collision detection algorithms as a wrapper.

For the aim of this project, the framework uses H3D's inherent collision detection approach without the complexity of generating a new algorithm to handle SOFA collision detection.

Appearance Model

Among the X3D specification settings, part of the design of the framework involves setting the values of the appearance model which must be suitable for the framework. An example of the settings most commonly used in the framework are shown as follows:

```
<Material
   ambientIntensity='1.0'
   diffuseColor='0.36 0.11 0.04'
   emissiveColor='0 0 0'
   shininess='0.2'
   specularColor='0.70 0.70 0.70'
   transparency='0' />
```

ambientIntensity defines the light that surrounds the object whereas *diffuseColor* defines the colour of the loaded virtual object. As the framework does not intend to use glow objects, *emissiveColor* field has been set to zero values. It is important to highlight as much as possible the geometry, so a low value on shininess is used. Additionally, the *specularColor* property is set to gray colour which properly highlights the view area without significantly altering the true colour.

transparency has been set to zero and by doing this, the rendering improves significantly. `Material` properties are chosen instead of an `ImageTexture` node because of better performance and aiming to see how shapes are deforming accordingly. Nonetheless, `ImageTexture` node is used in the surrounding organs, as shown in the spine and the diaphragm images in Figure 14 below:

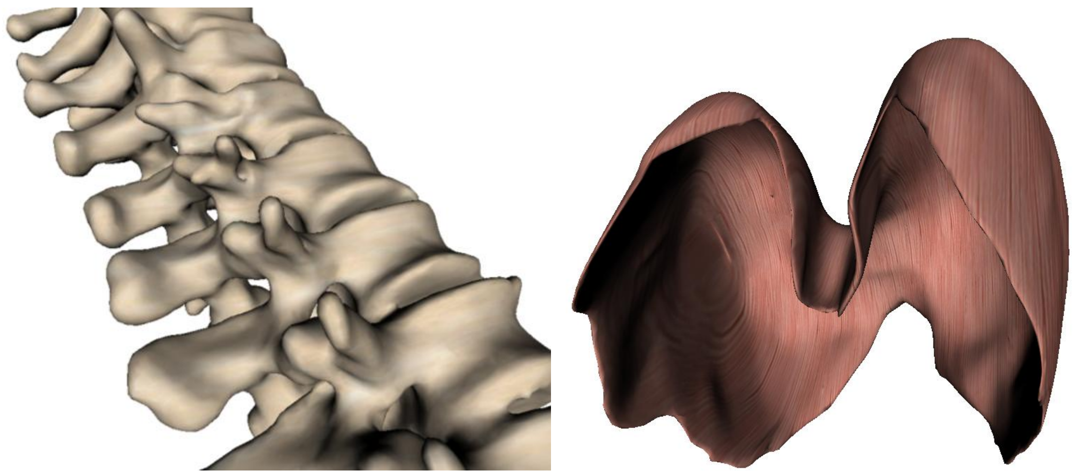


Figure 14 - Spine and Diaphragm with ImageTexture nodes

Haptic Shapes

From the haptics perspective, the geometry of the virtual object should only contain surface/haptic shapes. They are created as a composed geometry node in X3D by means of `IndexedTriangleSet` node. The geometry topology is an input into the framework and deals merely with triangle shapes.

There exists other ways to define a composed geometry in H3D as `IndexedFacedSet` does. However, the `IndexedTriangleSet` is used instead because triangles are defined as primitive shapes and collision objects in H3D and the surface faces of the virtual object from the volumetric mesh are triangles. Refer to Figure 15 as an example.

Haptics Global Settings

In H3D, global settings can be customised for the entire scene-graph. Three options were tested and their values depend on the user needs.

In the following XML tags there are recommended options from a performance concerning point of view. Broadly speaking, the use of bounding trees for collision detection has been disabled; the virtual object is only touchable from outside so once the avatar falls into the object can easily be taken out; and debug options are turned off which mainly draw bounding boxes and haptic shapes while in contact.

```
<GlobalSettings>
    <HapticsOptions touchableFace="FRONT" useBoundTree="false" />
    <GeometryBoundTreeOptions boundType="AABB"
                              maxTrianglesInLeaf="1" />
    <DebugOptions drawBound="false" drawBoundTree="-1"
                  drawHapticTriangles="false" />
</GlobalSettings>
```

Please refer to the Evaluation chapter to read about the tests carried out and the justification of the recommended settings within the framework.

Deformable Shapes Modelling

Geometry

Besides the surface geometry topology found for the haptics rendering, a mesh file consisting of a set of points and tetrahedra define the volumetric geometry for the deformation modelling. This file is an input to the framework and data is fetched from this file using a `MeshTopology` component in SOFA's scene-graph.

```
<Object type="MeshTopology" filename="liver.msh"/>
```

By using volumetric information in the deformation modelling the mechanical behaviour behaves realistically although the time integration is more computationally expensive as each node's position is calculated in each time step.

An example of a liver mesh with a cutting plane is shown in Figure 15.

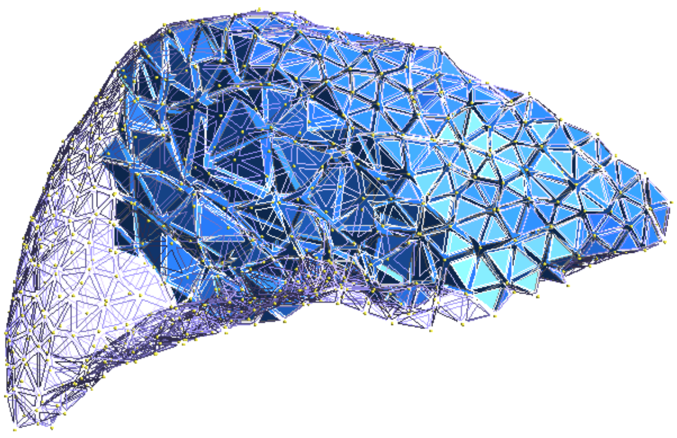


Figure 15 - Volumetric Mesh of a Liver

As it could be observed in this cutting-plane image, tetrahedra elements constitute the basis for the deformation modelling which will be integrated throughout time simulation.

Dynamic Modelling

Two different models available in SOFA using physically-based techniques for the deformation modelling of virtual objects were tested and plotted: `MeshSpring` and `TetrahedronFEM` force fields.

On the one hand, `MeshSpring` is a discrete model based on mass, springs and dampers to represent points, their interconnections and their resistance to motion respectively. On the other hand, `TetrahedronFEM` is a continuum model which uses tetrahedra as finite elements in order to approximate a solution by reducing partial differential equations into discrete algebraic equations.

Even when the user of the framework can choose any of the available methods, MeshSpring force field was selected due to its performance over an approximated physical properties object in aims for a real-time simulation with constrained hardware resources. Refer to the tests and results section of this document to see how both techniques performed using different mesh topology sizes.

Numerical Integration

One of the main elements of the framework is to find a solver which reaches equilibrium in real-time using large time steps for deformable bodies. SOFA has available several solvers which include `Euler`, `RungeKutta`, `Static` and `CGImplicit` solvers.

`CGImplicit` solver with a maximum of seven iterations is selected as the best solution for the tested resolution meshes. It becomes a good solution using large time steps, efficient animation and convergence.

In the analysis chapter solvers were studied according to the literature and the analysis support the results done in the evaluation chapter. Refer to both chapter for more information about the numerical integration solver.

Deformation Algorithm

Deformation algorithm refers mainly to the piece of code responsible to update the new point's position of a deformable shape. As described earlier, this algorithm executes the simulation loop and an X3D node was created under the name "Motion" and used as follows:

X3D node specification: `<Motion />`

Broadly speaking, the tasks performed in the algorithm are the following:

- As plasticity is not a property in human organs, the resting points keep the same throughout the simulation, thus the vector is cleared.
- Detect when the object is touched by iterating over all the connected haptic devices.
- If touched, send touch information to the GUI and console
- Then select closest vertices or closest vertex
- Send forces SOFA to identified vertex/vertices.
- Animate the mechanical behaviour
- Finally update new deformable points' position into H3D.

By selecting the closest vertex from the touched point, the perception of touching the object becomes more realistic especially for large topologies.

Framework Implementation

Introduction

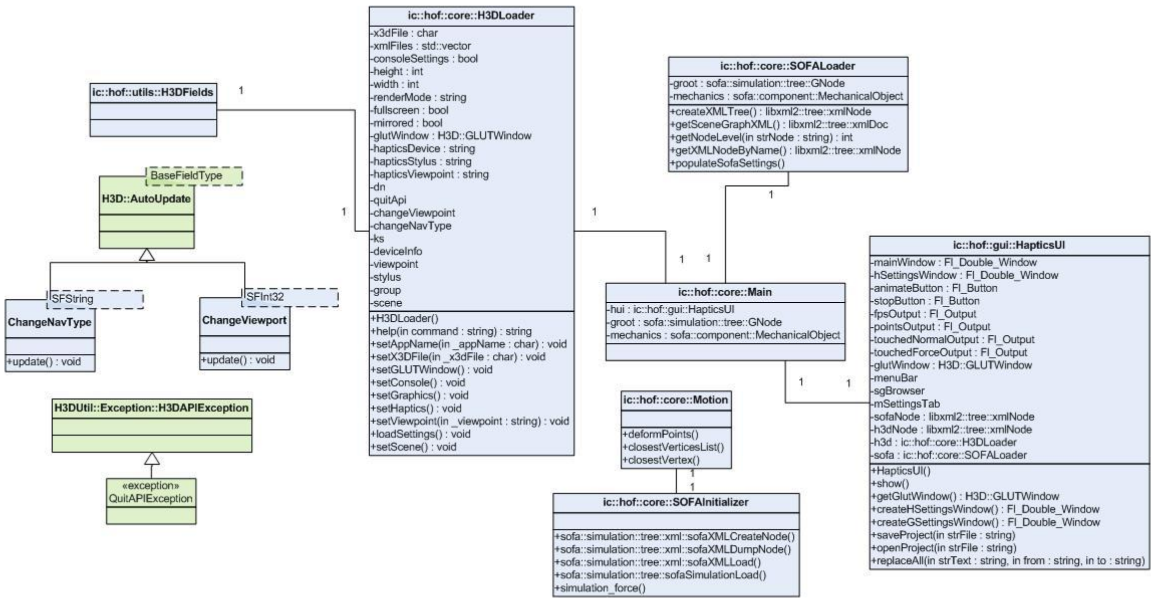
Objective

This chapter contains relevant information about the implementation of the Haptics Open Framework. Such information appears both as a background and as additional information of interest to follow up from the design chapter. Firstly, a more detailed overview of the software architecture is described as a specification perspective. Thereafter, more information of the source tool integration appears as well as some background knowledge to explain such integrations. Finally, a description of needle insertion tasks, the importance of simulating them and a couple of models as well as an implementation approach into H3D and the framework appears.

Software Architecture

A Specification Perspective

The diagram in Figure 16 shows the most important elements in the framework architecture. In the centre of the diagram the `Main` component mainly handles the global variables for the simulation, that is the graphical user interface (`hui`), the root of SOFA's scene-graph which will be animated each loop (`groot`) and the mechanical object as a reference for the points' position (`mechanics`). Linked to the `Main` component appears `HapticsUI`, `H3DLoader`, and `SOFALoader` as described in the design chapter.



ic::hof::utils::H3DFields
BaseFieldType
H3D::AutoUpdate
SFString
ChangeNavType
+update() : void
SFInt32
ChangeViewport
+update() : void
H3DUtil::Exception::H3DAPIException
«exception»
QuitAPIException
ic::hof::core::H3DLoader
-x3dFile : char
-xmlFiles : std::vector
-consoleSettings : bool
-height : int
-width : int
-renderMode : string
-fullscreen : bool
-mirrored : bool
-glutWindow : H3D::GLUTWindow
-hapticsDevice : string
-hapticsStylus : string
-hapticsViewpoint : string
-dn
-quitApi
-changeViewpoint
-changeNavType
-ks
-deviceInfo
-viewpoint
-stylus
-group
-scene
+H3DLoader()
+help(in command : string) : string
+setAppName(in _appName : char) : void
+setX3DFile(in _x3dFile : char) : void
+setGLUTWindow() : void
+setConsole() : void
+setGraphics() : void
+setHaptics() : void
+setViewpoint(in _viewpoint : string) : void
+loadSettings() : void
+setScene() : void
ic::hof::core::SOFALoader
-groot : sofa::simulation::tree::GNode
-mechanics : sofa::component::MechanicalObject
+createXMLTree() : libxml2::tree::xmlNode
+getSceneGraphXML() : libxml2::tree::xmlDoc
+getNodeLevel(in strNode : string) : int
+getXMLNodeByName() : libxml2::tree::xmlNode
+populateSofaSettings()
ic::hof::core::Main
-hui : ic::hof::gui::HapticsUI
-groot : sofa::simulation::tree::GNode
-mechanics : sofa::component::MechanicalObject
ic::hof::core::Motion
+deformPoints()
+closestVerticesList()
+closestVertex()
ic::hof::core::SOFAInitializer
+sofa::simulation::tree::xml::sofaXMLCreateNode()
+sofa::simulation::tree::xml::sofaXMLDumpNode()
+sofa::simulation::tree::xml::sofaXMLLoad()
+sofa::simulation::tree::sofaSimulationLoad()
+simulation_force()
ic::hof::gui::HapticsUI
-mainWindow : Fl_Double_Window
-hSettingsWindow : Fl_Double_Window
-animateButton : Fl_Button
-stopButton : Fl_Button
-fpsOutput : Fl_Output
-pointsOutput : Fl_Output
-touchedNormalOutput : Fl_Output
-touchedForceOutput : Fl_Output
-glutWindow : H3D::GLUTWindow
-menuBar
-sgBrowser
-mSettingsTab
-sofaNode : libxml2::tree::xmlNode
-h3dNode : libxml2::tree::xmlNode
-h3d : ic::hof::core::H3DLoader
-sofa : ic::hof::core::SOFALoader
+HapticsUI()
+show()
+getGlutWindow() : H3D::GLUTWindow
+createHSettingsWindow() : Fl_Double_Window
+createGSettingsWindow() : Fl_Double_Window
+saveProject(in strFile : string)
+openProject(in strFile : string)
+replaceAll(in strText : string, in from : string, in to : string)
1
1
1
1
1
1
1
1
1
1
1

Figure 16 - HoF UML Diagram

Additionally, in the left of the diagram, there are classes for handling the updates and routing of some H3D fields such as viewpoints and navigation types which change based on keyboard events.

`Motion` represents the incorporated node to handle the deformation of shapes within H3D. In this module, the deformation algorithm has been coded overriding the `deformPoints()` method, searches for the closest vertex given a touch point and send the forces to SOFA by means of calling `simulation_force()` in `SOFAInitializer` module which is represented as `Mechanics` in the conceptual perspective of the design chapter.

Open Source Tools Integration

FLTK – Graphical User Interface

Introduction

The FLTK integration has been carried out within H3D. The original H3D version uses Freeglut and wxWindows as part of its GUI. Freeglut is mainly used for the GLUT functions and wxWindows is used as a GUI in H3DViewer project. A brief description of the tools used by H3D is found below:

GLEW

The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. GLEW is used to determine which extensions are supported on the target platform. Currently it supports OpenGL, WGL and GLX extensions.

Once a valid OpenGL rendering context has been created, `glewInit()` function is called to initialize the extension entry points. GLEW_OK should be returned by this function. Checking for a particular extension is done by querying globally defined variables of the form `GLEW_{extension_name}`

GLU

Used by Freeglut, the GL Utilities (GLU) library is a set of routines designed to complement OpenGL by providing support for mipmapping, matrix manipulation, polygon tessellation, quadrics, NURBS, and error handling. An OpenGL context should be set before calling any GLU functions as they may call OpenGL routines. GLU routines are of the form `glu{routine_name}`

- Mipmapping routines include image scaling and automatic mipmap generation.
- Matrix manipulation functions build projection and viewing matrices as well as project vertices from one coordinate system to another.
- Polygon tessellation routines convert concave polygons into triangles for easy rendering.
- Quadrics functions render basic quadrics such as spheres and cones.
- NURBS routines map NURBS curves and trimmed surfaces into simpler OpenGL evaluators.
- Error handling routines translate OpenGL and GLU error codes into strings.

GLUT and OpenGL

The OpenGL Utility Toolkit (GLUT) is a window system independent API for writing OpenGL programs. GLUT is used for developing simple OpenGL applications. It is regarded as a simpler and easier approach for learning OpenGL programming. OpenGL is window system independent and the window system operations such as the creation of a window and the handling of window system events are left to the native window system to be defined. Instead of binding native window system APIs to OpenGL, GLUT provides a portable, window system independent and simple toolkit to develop programs. However, GLUT is not open source and there are available open-source alternatives such as Freeglut and FLTK, among others.

GLUT currently supports the following functionality:

- Multiple windows for OpenGL rendering
- Callback driven event processing
- Sophisticated input devices
- An idle routine and timers
- A simple, cascading pop-up menu facility
- Utility routines to generate solid and wireframe objects
- Support for bitmap and stroke fonts
- Miscellaneous window management functions

Additionally, GLUT routines are logically organized into several sub-APIs depending on their functionality. Below is a list of those subcategories used by H3D:

- Initialization (`glutInit`, `glutInitWindowPosition`, `glutInitWindowSize`, `glutInitDisplayMode`)
- Beginning event processing (`glutMainLoop`)
- Window management (`glutCreateWindow`, `glutSwapBuffers`)
- Callback registration (`glutDisplayFunc`, `glutKeyboardFunc`, `glutMouseFunc`, `glutIdleFunc`)

Freeglut

Freeglut is an open-source clone of the GLUT library which allows the user to create and manage windows containing OpenGL contexts. It can be used either as a static or as a dynamic library (DLL).

`H3D::GLUTWindow` can be regarded as a wrapper containing encapsulated GLUT functions. `H3D::GLUTWindow` is a child of `H3D::H3DWindowNode` abstract class which contains all the information about the current window as a node. `H3D::Scene` class uses this window to render the set of scenes in H3D. Both `H3D::X3DKeyDeviceSensorNode` and `H3D::MouseSensor` classes inherits from `H3D::X3DSensorNode` class. All these classes are inherited by `H3D::Node` abstract base class.

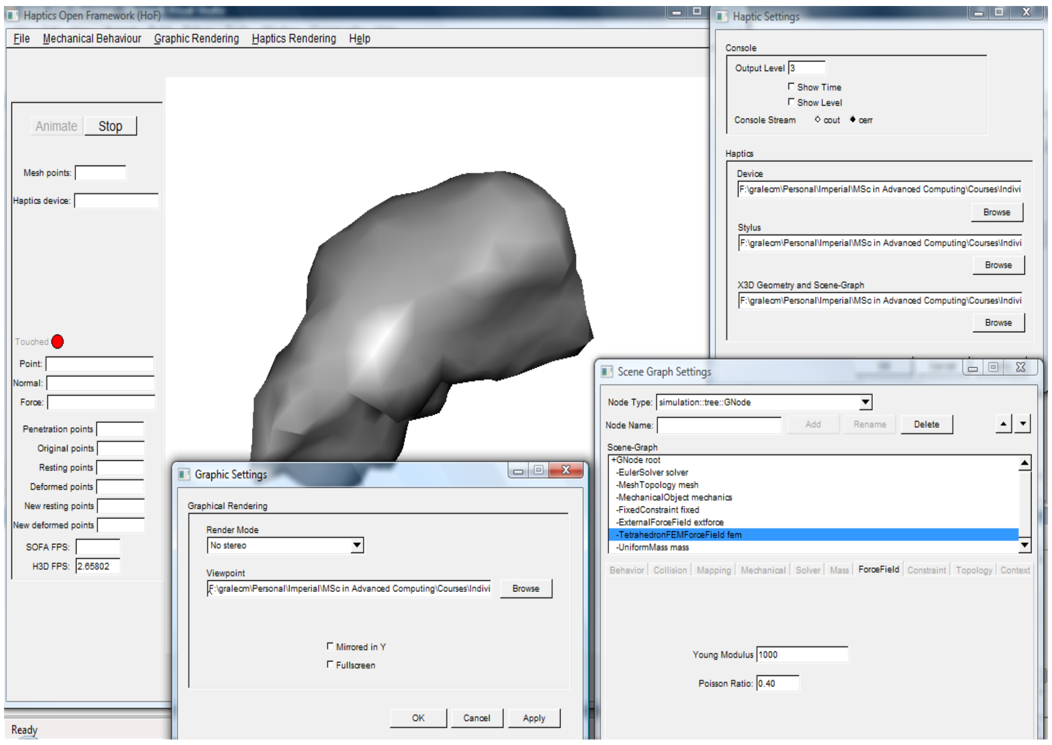
H3DAPI Source Code Modifications

A window in H3D is represented as a node. H3D::GLUTWindow is a GLUT implementation of `H3D::H3DWindowNode` using Freeglut. As Freeglut has been replaced by FLTK, this class needs to extend FLTK `Fl_Window` class in a similar approach as `Fl_Gl_Window` is extended for OpenGL inside FLTK. Refer to Figure 18 to see the inheritance diagram of H3D::GLUTWindow.

By extending `Fl_Window`, `H3D::GLUTWindow` will be used as a subwindow embedded into the GUI which parent window is the one containing all the widgets. Please refer to the appendix for detailed information on H3DAPI source code modifications.

GUI Components

Figure 17 shows a screenshot of some of the widgets and windows used in the framework. FLUID was used as a GUI editor to build the graphical user interface and connect the widget objects to each of the modules in the framework.



H3D FPS: 2.65802

Render Mode: No stereo

Viewpoint: F:\gralecm\Personal\Imperial\MSc in Advanced Computing\Courses\Indivi

Node Type: simulation::tree::GNode

```
+GNode root
  -EulerSolver solver
  -MeshTopology mesh
  -MechanicalObject mechanics
  -FixedConstraint fixed
  -ExternalForceField extforce
  -TetrahedronFEMForceField fem
  -UniformMass mass
```

Young Modulus: 1000

Poisson Ratio: 0.40

Output Level: 3

Console Stream: cout [✓ cerr]

Device: F:\gralecm\Personal\Imperial\MSc in Advanced Computing\Courses\Indivi

Stylus: F:\gralecm\Personal\Imperial\MSc in Advanced Computing\Courses\Indivi

X3D Geometry and Scene-Graph: F:\gralecm\Personal\Imperial\MSc in Advanced Computing\Courses\Indivi

Figure 17 - GUI components

From left to right, the main window appears in the background with the animate and stop buttons which handle the animation. At the top of this main window a menu bar has been set up. Additionally on its left, information such as points, normals and forces are displayed when the object is in touch.

The first sub window is used for the graphics settings. The render mode can be selected from a list and the viewpoint of the scene graph is selected by browsing a file which contains a Viewpoint node with position information as shown below:

```
<Viewpoint position="0 0 0.6"/>
```

The next sub window has been designed for the mechanical behaviour settings. A browser widget has been configured to create and modify SOFA scene-graph. A tree can be built in this browser by selecting the type of component/node and their properties are customised in several tabs which appear below the browser.

Finally, the last sub window which appears in the right of Figure 17 is used to configure the haptics settings. Console settings can be customised and the user can browse files to specify the device info node, the stylus representation and the surface topology. All these files are in X3D format and an example of the device info file for the Falcon device is shown below:

```
<DeviceInfo>
  <FalconDevice positionCalibration="3 0 0 0
                                     0 3 0 0
                                     0 0 3 0
                                     0 0 0 1" >

    <RuspiniRenderer/>
  </FalconDevice>
</DeviceInfo>
```

There is an example of a stylus for representing a needle in the appendix as well as an example of how the topology node is defined.

H3D – Haptics Rendering

H3DAPI provides an implementation of the X3D specification by defining nodes and their updates as classes and fields as attributes mainly under `H3D::X3DNode`.

It also serves as a wrapper for HAPI to define forces, shapes, haptic devices and haptic rendering algorithms among others so the user can create a scene-graph by means of using nodes and fields.

The most important component is H3D::Scene class which executes the main loop and traverses the scene-graph. Figure 18 shows the main components in H3D and their relationships.

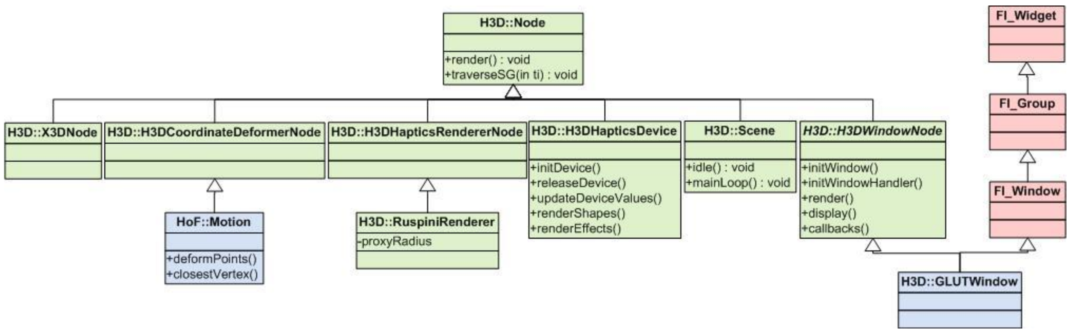


Figure 18 - H3D & FLTK UML Integration

As it can be seen, H3D::Node is the main class which classes inherits from. Nodes could be ranging from those found in X3D specification, coordinate deformers for deformable shapes, haptic rendering nodes, haptic device nodes, to even the scene and the windows.

In this particular diagram, it can be observed how GLUTWindow has been customised for the FLTK integration. Additionally it shows how the deformation algorithm described earlier in the design chapter is connected to H3D.

SOFA – Biomechanical Simulation

The main components in SOFA that interact with the Haptics Open Framework are shown in Figure 19.

`GNode` is used as a reference to the main node in SOFA's scene-graph. It also has a `Context` where the time step is set for the whole scene. Every time the deformation loop in `Motion` executes, SOFA is animated using `GNode`. Shortly thereafter, a second reference to `MechanicalObject` has the element's new position which is fetched by calling its `getX()` method.

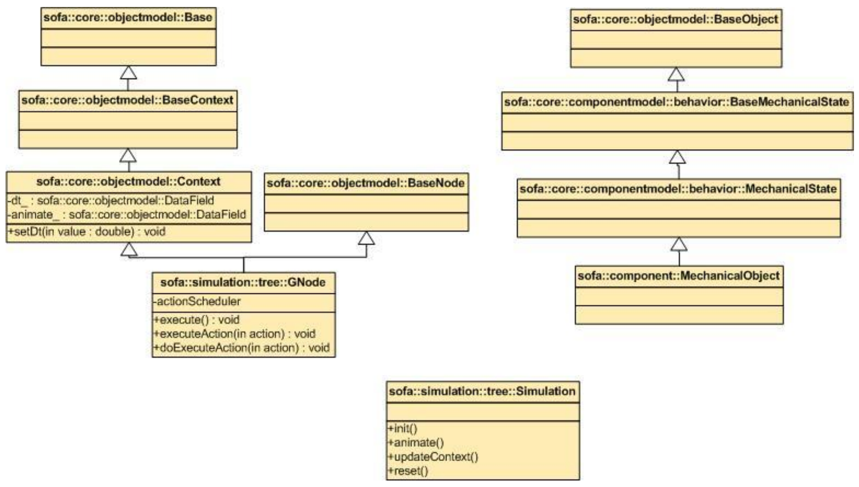

sofa::core::objectmodel::Base
sofa::core::objectmodel::BaseContext
sofa::core::objectmodel::Context
-dt_ : sofa::core::objectmodel::DataField
-animate_ : sofa::core::objectmodel::DataField
+setDt(in value : double) : void
sofa::core::objectmodel::BaseNode
sofa::simulation::tree::GNode
-actionScheduler
+execute() : void
+executeAction(in action) : void
+doExecuteAction(in action) : void
sofa::core::objectmodel::BaseObject
sofa::core::componentmodel::behavior::BaseMechanicalState
sofa::core::componentmodel::behavior::MechanicalState
sofa::component::MechanicalObject
sofa::simulation::tree::Simulation
+init()
+animate()
+updateContext()
+reset()

Figure 19 - SOFA UML Integration

Regarding to the mechanical behaviour scene-graph, once the user has either created a new one or load a project, the information contained in the FLTK widgets is used to create an XML tree document using Libxml. This XML document is used to load the scene-graph settings into SOFA simulation by parsing the tree looking for nodes, objects and their attributes.

Even though functions for XML parsing are provided by SOFA, some of them were adapted to the framework because SOFA loads a file instead of an XML document. As a consequence, the user not only can create the scene-graph by using graphical components but also can modify the settings online during animation which makes the framework more useful.

OpenMP – Optimisation

OpenMP was tested in the framework for the update of the new points' position from SOFA mechanics to H3D deformed points. A for loop directive was used as shown in the code below.

Shared variables are listed and correspond to those elements which will be accessed by all team members, that is, by all the threads in the parallel region whereas private variables are specific to each thread. The following code was copied from the deformation algorithm and shows how OpenMP was configured and adapted for updating the position values.

```
#include <omp.h>

…

#pragma omp parallel
       shared(mechanics,new_deformed_points,total,chunk)
       private(i,x,y,z,node)
{
   #pragma omp for schedule(dynamic,chunk) nowait
   for( i=0; i<total;i++ ) {
         x=(float) (*mechanics->getX())[i][0];
         y=(float) (*mechanics->getX())[i][1];
         z=(float) (*mechanics->getX())[i][2];

         node[0]=x; node[1]=y; node[2]=z;
         new_deformed_points[i] = node;
   }
}
```

The schedule clause describe how the for loop is divided by the members of the team. Dynamic schedule divides the loop iterations in pieces of chunk size and shortly thereafter they are dynamically scheduled among the threads. The chunk value was set to a value of 50 thinking about iterations for large topology sizes. There is no need to synchronise at the end of the loop so `nowait` has been set.

Needle Insertion

Introduction

The purpose of this section is to describe the importance of simulating needle insertion procedures in medical applications and to provide current literature approaches for modelling it. Furthermore, an explanation of how Fung model was designed and implemented into the haptics open framework appears at the end of this section.

Needle insertions have been used in many traditional medical treatments such as injections and biopsy. These methods are among the less invasive procedures in radiological and surgical techniques. It is expected that these techniques increase during the following years as pointed out by Maurin et al [2004].

Needle insertion procedures require both the visual feedback and the haptic feedback. The first one is useful for the position of the inserting point and penetration whereas the second one provides great source of information for practitioners. Percutaneous procedures require accuracy for targeting organs which is achieved by the combination of imaging and haptics.

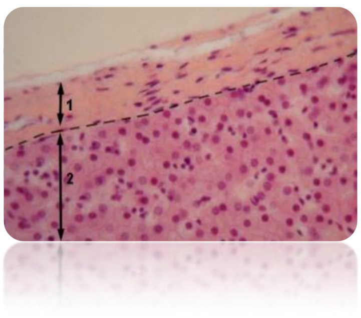
Nienhuys et al [2003] identify two main needle insertion procedures:

- Needles with a special bevel have to be inserted deeply to extract a tissue specimen as done in biopsies.
- Brachytherapy is another application for cancer treatment where radioactive seeds are inserted directly into the tumour

Due to deformation of soft tissue, needle insertion becomes a difficult task in order to reach the desired target and avoid vital organs. Simulations of needle insertion may help train and plan for these procedures.

Particularly for the liver, Zamani et al [2007] state that this organ may deform largely under the influence of factors such as gravity, cancer and position of needle, among others. Additionally, the liver is surrounded by a thin membrane called capsule which presents another challenge in its nature for the needle insertion simulation. Figure 20 shows a microscopic histology image of the liver with both the capsule and the liver parenchyma.

Figure 20 - Microscopic Histology Image of the Liver



In Maurin et al paper [2004], several experiments were carried out for both manual and robotic insertions. They identify three phases in a needle insertion: insertion, relaxation and extraction. In the insertion phase, forces behave as a succession of exponential-like rise before and after the liver capsule rupture. The maximum force is felt when movement ceases and it is proportional to the depth of penetration. During the relaxation phase, a slow decreasing repulsion force is applied along the axis of the needle even though it is not moving. Such force depends on the bevel of the needle. During extraction, friction forces are applied on the needle surface. An exponential-like decrease appears as in the insertion phase after reaching 0N. Forces experimented in each phase are plotted in Figure 21.

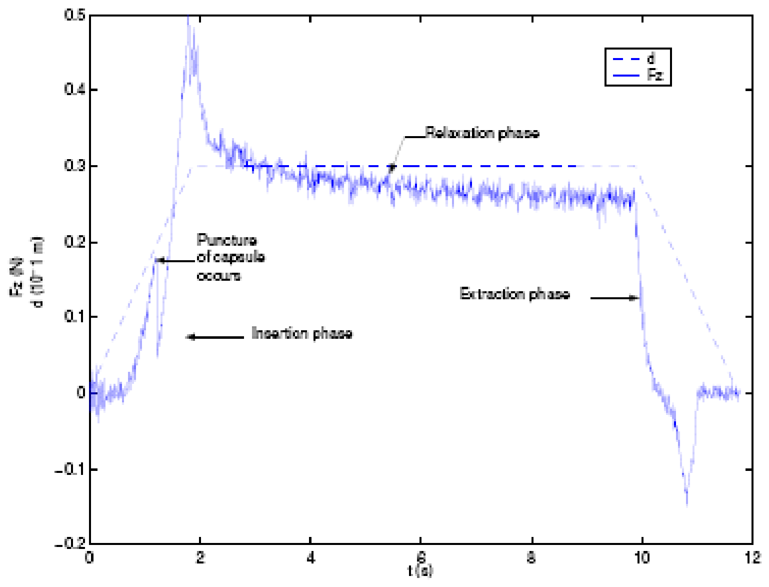


Figure 21 - Force Response during Needle Insertion into the Liver

Maurin et al [2004] modelled a needle insertion procedure into the liver in two different approaches with low error rate.

- Adapted Simone and Okamura model
  This model includes stiffness and friction forces only. Stiffness force is related to depth with a second order polynomial with coefficient values a_0=0N, a_1=0.002N/mm, and a_2=0.0023N/mm2

$$f(d) = a_0 + a_1 d + a_2 d^2$$

  Friction force is given by a Karnopp model with static friction values of 0.121mN/mm and -0.117 mN/mm were obtained, whereas damping coefficients were 3.2mNs/mm2 and -1.6mNs/mm2.

- Maurin approach based on Fung model
  This model considers two phases, before and after capsule puncture. In each phase, the force is modelled as a function of depth represented as follows, theta = [a,b,d0,F0]:

$$f(d) = (F_0 + b)e^{a(d-d_0)} + b$$
$$\theta = [a, b, d_0, F_0]$$

For the first phase θ=[0.121, -0.098, 11.45, 0.2] and for the second phase θ=[-0.031, 1.7, 19.61, -3.39] were found as parametric vector values.

In the following two sections, a description about the design and implementation of needle insertion in H3D is shown.

Design

One of the key features is to extend the Haptics Open Framework to simulate needle insertion procedures. In order to model a procedure of this kind, the Fung's model approach described in Maurin et al [2004] is used for a human liver. Additionally, the modelling of the liver capsule becomes a challenge as the framework process only the surface of a virtual object, even though the volume is considered merely for deformation.

An initial approach for the modelling of the capsule considered haptic layers which are provided by H3D. The liver surface and its capsule are then represented in different layers. Depending on the layer the proxy is laying at, forces are computed based on the previous models described by Maurin et al [2004]. Figure 22 show how the needle tip can be rendered in two different layers.

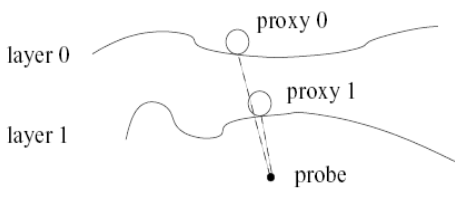


Figure 22 - Haptic Layers in H3D

However, the haptics device will receive a resulting force which is generated by adding up all the forces computed in each layer. This is not adequate as we need the forces of each layer to be sent independently to the haptics device depending in which layer the tip of the needle is at. Also, a proxy-based haptic rendering algorithm will always keep the proxy along the surface and a second geometry file for the capsule needs to be provided.

A second approach is to keep on using only one layer and to generate a force. There are two different ways to control the amount of force that is sent to the haptics device, either haptic surfaces or force effects. Both approaches are used for the modelling and the implementation is described in the next section.

Implementation

A new `H3DSurfaceNode` called `LiverSurface` is created along with its counterpart `HAPISurfaceObject`. The main purpose is to generate a zero valued force once the haptic surface is in contact. By doing this, the proxy and probe's positions are not connected by any force spring and the distance between them will be computed as the penetration depth of the needle's tip. Such surface node can be defined for the `DeformableShape` in X3D as follows:

```
<LiverSurface />
```

This surface node works as a wrapper for the surface object defined in HAPI which is called `HAPILiverSurface`. A surface object defines the haptic properties of a geometric shape such as stiffness and friction. It is responsible for generating forces at a local contact point. As a child of `HAPISurfaceObject`, two methods need to be overridden. The first one called `getProxyMovement()` is used to control the feeling of the object while moving across the surface. It has been configured exactly the same as a smooth surface which is represented by minimising the distance between the probe and the proxy and move along the contact plane, so no tangential forces are applied. The second method called `getForces()` computes the interaction force once the new position of the proxy has been calculated. This function is used to always set the contact force to a zero value.

Thereafter, the only forces exerted into the haptics device will be those computed by a new node in the form of an `H3DForceEffect` once a penetration depth exists. By means of using some fields to represent when the surface is touched and when the capsule has been broken, then a resulting force is computed based on Maurin et al [2004] work. This node can be specified in X3D as follows:

```
<LiverFungForceEffect DEF="FUNG" capsule="0.02" />
```

This force effect node is defined along with six fields listed below and which routing has been configured in Figure 23. In X3D, fields are used as event handling mechanisms and are arranged in a field network by means of routes which convey these events throughout the network.

- *touched* (`SFBool`) is used as a flag and determines when the object is in contact.
- *capsule* (`SFFloat`) specifies the distance from the surface to the liver capsule and helps as a threshold to indicate that the needle tip has penetrated it.
- *broken* (`SFBool`) is used as a flag to indicate when the liver capsule has been broken.
- *distance* (`SFFloat`) represents the penetration depth distance which is computed using touch point and penetration point in `Motion` node.
- *force* (`SFVec3f`) is the computed force based on Fung model which is computed with two different θ values depending of the phase, that is, before or after the liver capsule rupture.
- *normal* (`SFVec3f`) is the force normal applied to the haptics device and is computed as the negative touch normal.

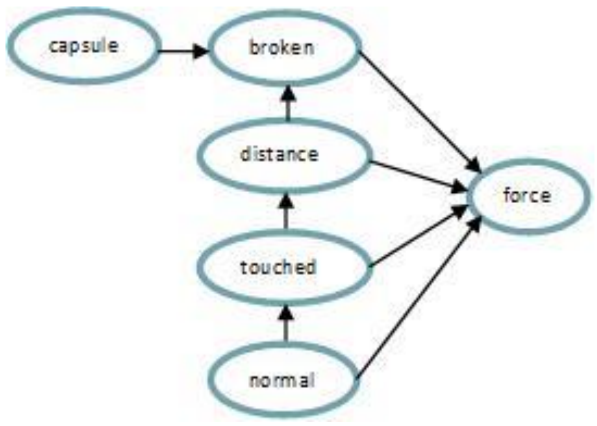


Figure 23 - Field Network for Needle Insertion

Both broken and force fields use the `TypedField` template modifier in order to update their values based on their incoming routes as seen in Figure 23. Touched, distance and normal fields are updated from `Motion` node which runs the simulation loop for deformable shapes. Routing is configured using C++ within the constructor of the force effect node class as follows:

```
capsule->route(broken);
distance->route(broken);
touched->route(force);
distance->route(force);
broken->route(force);
normal->route(force);
```

Also, the deforming algorithm in Motion has been modified to access LiverFungForceEffect and to print relevant information, as well as sending the computed force to SOFA. Therefore a new node was created and can be specified as follows for the deformable shape.

```
<MotionNeedle />
```

Finally, a simple geometric representation of a syringe and a needle is represented for the stylus as shown in Figure 24.

Figure 24 - Virtual Representation of a Needle in H3D

The needle has been created based on the needle tip characteristics described in Zamani et al [2007] where the diameter of the needle is 1.25mm, its tip diameter measures 0.25mm and the total length is 94mm. For simplicity, the needle has been divided into two sections of same length, that is, 47mm each. The first section represents the tip's diameter whereas the second represents the needle's diameter. The appendix of this document includes this X3D representation for more detailed information.

Haptics & Deformation Analysis

Introduction

Objective

This chapter contains a link between implementation and literature in the two most important libraries in the framework, that is, SOFA and H3D. Additionally, analysis of some features is described in a way to compare functionality as in the case of collision detection algorithms and at the same time provide a fundament from the implementation point of view. The reader might use this chapter as a reference to other chapters such as Design and Evaluation.

Haptic Rendering

Haptic Rendering Algorithm

A haptic display is the process of applying forces in order to give the sensation of touching things in an environment. Zilles et al [1995] describe a haptic display composed by three main components: a) a haptic interface/device which is an electro-mechanical system able to send forces to the user with one or more DOF, b) an object model which provides a mathematical representation, and c) a haptic rendering algorithm which uses the last two components to compute forces in real time to give the sensation of touching.

There are four haptic rendering algorithms available in H3D which are configured by means of the X3D scene-graph DeviceInfo node:

```
<DeviceInfo>
   <FalconDevice positionCalibration="…">
      <…Renderer/>
   </FalconDevice>
</DeviceInfo>
```

GodObject renderer

Based on the research done by Zilles and Salisbury [1995], it uses a point proxy. Among the advantages, it is open source, device independent, and supports user defined surfaces. However, fallthrough problems appear on moving objects.

The point interaction method simplifies both device and algorithm development as just the point tip of the avatar is rendered. Accordingly to Zilles et al [1995], most users tend to use less than 5N (1lbf) of force in virtual environments. At the time, previous haptic rendering algorithms had been grouped together as vector field methods which determine the force based on the penetration depth. This approach led to some disadvantages such as an unclear association of surface to internal volume, force discontinuities while traversing volume boundaries, and thin objects.

God-object is the virtual representation of the haptic interface and always remains on the surface of the objects whereas the haptic interface point does the penetration of the object as shown in Figure 25 in green and blue colours respectively. In free space they share the same position. The God-object algorithm basically works as follows: 1) previous and current servo cycle god-object’s position define the surfaces touched, 2) Lagrange multipliers are used to find the new location during contact which is the minimum distance between the god-object and the haptic interface point subject to a particular surface, 3) then stiffness and damping can be applied between the haptics interface point and the god-object to compute the forces and represent local properties.

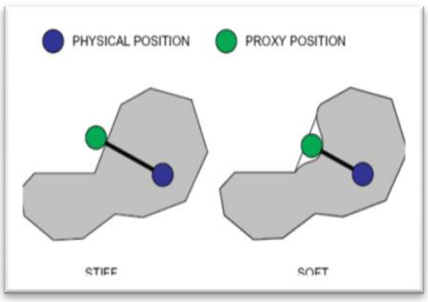


Figure 25 - Proxy-based Haptic Rendering

Ziles et al [1995] manifest some constraints in the algorithm which include: a) a mesh of triangular elements in the surfaces is used as it is the most fundamental and assures all the nodes are coplanar, b) fallthrough problems are caused due to round-off errors in the newly computed god-object position, and c) for convex portions of objects, only one surface is active at a time.

X3D node definition: `<GodObjectRenderer />`

Ruspini renderer

Based on the work by Ruspini et al [1997], it uses a variable sphere proxy size. It is open source, device independent, and supports user defined surfaces, as God-Object renderer. However, it is slightly slower than others.

Ruspini et al [1997] based the idea of constrained-based methods such as the god-object algorithm to introduce force shading, friction, surface stiffness, and texture by merely changing the position of the proxy. The radius of the proxy is changed to overcome the problem of falling through the object in a dynamic environment. Due to small numerical errors, polygons that share a common edge tend to leave small gaps. Therefore, the radius of the proxy should be large enough.

Ruspini devise the idea of configuration-space obstacles which are defined consisting of all the points within one proxy radius from the object surface. Then, constrained planes are computed depending on which element is in contact: none, one, two or three plane constraints for free space, a point, an edge or a vertex respectively.

Force shading is analysed in Ruspini's paper even though it is not implemented in H3D. It is based on Gouraud and Phong for a lighting model where interpolated surface normals can be applied to vertices. The same effect can be applied to haptics by changing the direction of the normal force and keeping its magnitude. First, a new plane constraint is defined by the interpolation of the vertices normal. This plane is called the force shading plane and goes through the point in contact. The second step is to find a sub-goal perpendicular to this plane based on the current finger position. This sub-goal eventually will become the new finger's position. Based on the finger's new position, a new sub-goal is defined but now depending upon the original constraint plane in order to find the new proxy goal position. Figure 26 shows the two steps performed.

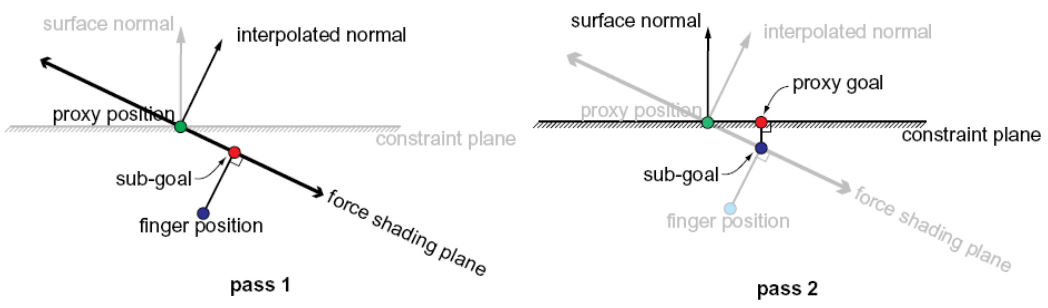


Figure 26 - Two Pass Force Shading

X3D node definition: `<RuspiniRenderer />`

Chai3D renderer

It is an open source library developed at Standford University. As GodObject and Ruspini, it is also device independent. However, user defined surfaces are not allowed and sometimes fallthrough problems exist on moving objects.

The implementation of Chai3D in H3D uses an implicit function approach to describe a geometric surface. Such functions are represented by mathematical definitions which make easier to identify collisions and its point of interaction.

Additionally, Chai3D does force shading which has been pointed out during the evaluation chapter. Mainly, interpolated surface normals have been defined in the vertices of an object to create perpendicular resulting force to any point the proxy is laying in the surface. Without force shading, strong discontinuities occur when the proxy reaches an edge. Both scenarios are shown in Figure 27.

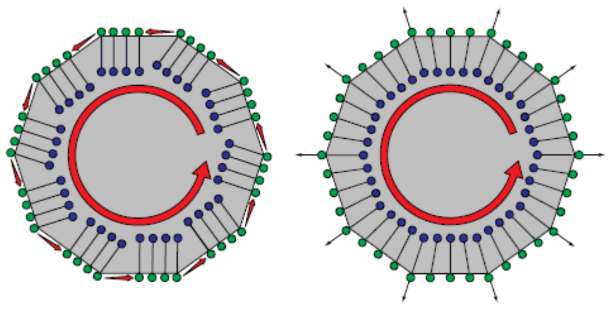


Figure 27 - Flat and Force Shaded Surfaces

X3D node definition: `<Chai3DRenderer />`

OpenHaptics renderer

OpenHaptics is a proprietary library developed by SensAble®. It uses a point proxy and provides stable haptic feedback. In H3D, there is a MagneticSurface available which performs fine with moving objects. Basically, the proxy is attracted by the surface and forces are created accordingly. However, the algorithm is device specific for SensAble® devices, no user defined surfaces are allowed, and it is not open source.

X3D node definition: `<OpenHapticsRenderer />`

Collision Detection Algorithms

H3D

Collision detection algorithm forms part of haptic rendering in H3D. The main purpose is to generate force and torque values to send to the haptics device given a set of geometries and current values of the device. Device values are represented in both local and world coordinates.

`HAPI::Bounds::CollisionObjects` is the base class for objects that can be used in collision detection. The inherited object classes implement the functions `closePoint()`, `lineIntersect()`, and `movingSphereIntersect()`.

There are three different types of object primitives: `BoundPrimitives`, `GeometryPrimitives` and `BinaryBoundTrees`. Bound primitives consist of `AABoxBound`, `OrientedBoxBound` and `SphereBound`. Geometry primitives include plane, sphere, triangle, line segment and points. Binary bounding trees can be `AABBTree`, `OBBTree` or `SphereBoundTree`.

SOFA

One of the main components in SOFA for collision detection is the CollisionPipeline. Basically, the pipeline resets previous loop, computes collision detection given a model and sends response which include colliding elements, points, normal and distance.

The second main component in SOFA is the collision model to use. A `CollisionModel` contains a list of same type elements or a list of collision models each describing a level in a bounding tree hierarchy either finer/lower/child or coarser/upper/parent.

As an example, a sphere collision model using `component::collision::Sphere` is defined in the scene-graph as follows:

```
<Object type="Sphere" name="CollisionModel" filename="liver.sph" />
```

Such model defines spheres as collision objects which properties such as centre and radius are defined by means of a SPH file. A graphical view of this representation is depicted in Figure 28:

```
sph 1.0
#nums 27
#   xmin="-5" xmax="2"
#   ymin="0.25" ymax="5.25"
#   zmin="-2" zmax="3"
sphe 1 -4 1.25 0 0.85
sphe 1 -4 1.25 1 0.85
…
```

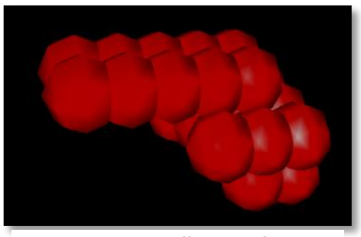


Figure 28 - Collision Object Representation of the Liver

X3D Specification in H3D

Introduction

This section partly describes how H3DAPI implements X3D specification for the scene-graph settings and some of the nodes that are used in Haptics Open Framework. Such specification is defined by X3D in part one called “Architecture and base components”. It contains the abstract functional specification for the X3D framework, and definitions of the standardized components and profiles.

Abstract Classes

`H3D::Node` is the base class for all the classes that can be part of the scene-graph. `render()` function is used for the traversal of the scene-graph and `traverseSG()` is called once per loop.

`H3D::X3DNode` is the base class for all the nodes in the X3D system and `H3D::X3DChildNode` abstract class handles X3D children fields. Concrete nodes derived from `H3D::X3DGroupingNode` abstract class contain children nodes and it is the basis of all aggregation. Rendering the group involves rendering all its children. In the same way, traversing the scene graph for the group involves traversing all its children.

X3D Nodes

Group

`H3D::Group` contains children nodes which are enclosed by a bounding box.

Transform

`H3D::Transform` grouping node defines a coordinate system for its children relative to its ancestors. A 3D point is transformed into another point in its parent coordinate system by a series of intermediate transformations.

DeformableShape

`H3D::DeformableShape` makes it possible to deform the geometry of the shape when touching it with a haptics device.

The deformer field defines an `H3DCoordinateDeformer` node which determines how the coordinates should be deformed on contact. `H3DCoordinateDeformer` has been used as the base class for `Motion` node, which is used by the Haptics Open Framework to communicate with SOFA. `H3D::H3DCoordinateDeformerNode` is the base class for all the nodes specifying deformation based on contact of haptic devices. Deformation is defined in the implementation of `deformPoints()` method.

The geometry field must contain an `X3DComposedGeometryNode`. The deformable coordinates are defined in its `coord` field.

Geometry

H3D::X3DGeometryNode is the base node for all geometry nodes in X3D. H3D::X3DComposedGeometryNode is the base node for all composed geometry in X3D. A composed geometry node composes geometry from a set of nodes that define individual components such as colour, normal and texture, among others.

IndexedTriangleSet

`IndexedTriangleSet` node is used to specify the geometry of a surface for the haptics rendering. This node is a subclass of `H3D::X3DComposedGeometry`. It specifies a 3D shape as a collection of individual triangles. Using its `Coordinate` fields it arranges every triangle by referring to its coordinate index.

Appearance

X3D defines equations for the lighting model which specifies the colours to apply to the geometry. The model takes into consideration the `Material` node along with values such as the distance from the viewer to the geometry and the distance form light to point geometry among others.

`Shape` nodes can associate geometry nodes to a particular appearance. An `Appearance` node defines properties such as material, texture and texture transformation to be applied to the geometry of the shape nodes. The `Material` node specifies surface material properties and is used by the X3D lighting model equations. The fields in the material node define how light reflects off the geometry to create colour.

Deformable Shapes Modelling

Numerical Integration Solver

`Euler` and `RungeKutta` methods use explicit time integration to approximate a solution in order to find element's position and velocity at time steps throughout the simulation. The smaller the time step the more accurate the approximation. Explicit methods are based on data already known as previous positions and velocities whereas implicit methods need to solve a system of equations.

The standard version of the Euler method is a first order numerical procedure for solving ordinary differential equations based on an initial value. The goal is to approximate a function by using the slope of the tangential line of a known point in the curve. Then, a small step is taken over the tangent line to result in a new point which is considered for the next iteration. If this process repeats several times, a polygonal approximated curve is generated. Therefore, the smaller the time step, the better the function approximation is. The method is explicit because the value computed in the previous iteration is needed to compute the current iteration value.

Symplectic Euler is a modified version of the standard version in the form of a semi-implicit integrator. It is applied to a pair of differential equations where one depends on the other as it is the case for displacement and velocity. It is called semi-implicit because in order to get the new particle's position it uses the current computed velocity instead of that one from the previous iteration.

RungeKutta4 is similar to the standard Euler method as an explicit method. However, the next point is computed by using a weighted average of slopes computed at the beginning, in the middle and at the end of the time step interval.

Static solver aims to find the static equilibrium of a system although can diverge when there are infinity of solutions. It uses a conjugate gradient method. On the other hand, `CGImplicit` is based on Baraff et al [1998] paper for cloth simulation. They use implicit integration with a fast iterative algorithm, that is, a modified version of conjugant gradient method. Additionally, the constraints are always the same regardless the number of iterations to solve the linear system. This proposed solution achieves larger time steps which complement the small number of iterations to converge. The use of implicit integration generates large sparse linear systems which are easily solved using conjugate gradient as they are based on matrix-vector multiplies.

Evaluation

Mesh Topology Sizes

Introduction

Haptic Open Framework has as inputs both volumetric data and surface data for the geometry definition of a virtual object. An ongoing project processes CT and MRI images by performing segmentation and meshing techniques and consequently produces a 3D model representation of that object.

Volumetric data is represented as a mesh file (*.msh) which is composed by a set of points and tetrahedra. Additionally, an X3D file (*.x3d) contains both Coordinate and `IndexedTriangleSet` nodes which are used to define the surface consisting of a set of points and triangles respectively. The point sets from both files match in its number of elements and coordinates, therefore a point in the surface in the volumetric representation matches the same index of the point in the surface representation.

Scenarios

Four different resolution levels of accuracy serve as input for a human liver. The number of points, tetrahedra and triangles are plotted in Figure 29. Each resolution has been labelled as: Liver Simple, Liver Coarse, Liver Medium and Liver Fine. Such labels will be used throughout the tests in this chapter and the reader can refer to them in this section.

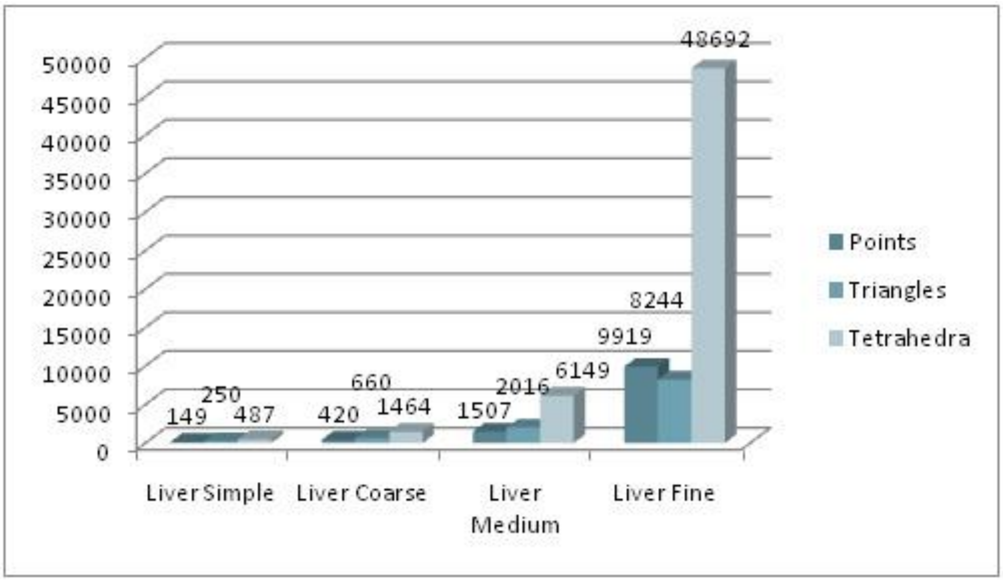


Figure 29 - Mesh Topology Resolutions

Haptic Rendering Algorithms

Introduction

Three different haptic rendering algorithms were tested using deformable shapes in the framework: God-object, Ruspini and Chai3D. OpenHaptics algorithm was not tested at all because of its disadvantages to the present work: closed source and device dependent.

Tests and Results

The test scenarios involved four different mesh topology sizes which were described previously. A CGImplicit solver with five iterations as maximum and a MeshSpringForceField were used for the numerical integration. Stiffness was set to a value of 5000 and damping to 25. When the surface is touched, just the closest vertex is sent as index together with the touched force to the ExternalForceField.

As an overall, GodOject is not a good solution for deformable shapes owing to many fallthrough problems. It was quite complicated to touch the surface and press it by any given force using the Falcon haptics device. On the other hand, Ruspini and Chai3D perform well in the tests even when both fall through the surface sometimes. In the case of Ruspini, these kind of problems exist when pushing an object and moving along a surface with sharp edges at the same time. Chai3D presented fewer problems than Ruspini in this context. Nonetheless, unstable force feedback behaviour is presented in Chai3D for all the mesh topology sizes.

Additionally, as surface normals are not interpolated, geometry edges are perceived by using the haptic device. This feeling decreases as the mesh topology size increases. By using Chai3D a smooth surface was perceived when moving along the surface.

These algorithms did not show any computing overhead compared against the others and Ruspini was more appropriate to the framework as an overall.

Haptics Global Settings

Introduction

H3D allows modifying option settings for the entire scene-graph. These options become very important for the framework and their values depend on user needs. Therefore some tests were carried out in order to present some recommendations. For an easier explanation, the same XML code from the design chapter has been copied in this section and appears below.

```xml
<GlobalSettings>
    <HapticsOptions touchableFace="FRONT" useBoundTree="false" />
    <GeometryBoundTreeOptions boundType="AABB"
                              maxTrianglesInLeaf="1" />
    <DebugOptions drawBound="false" drawBoundTree="-1"
                  drawHapticTriangles="false" />
</GlobalSettings>
```

Tests and Results

In HapticsOptions, *touchableFace* becomes very important due to the fallthrough problems found in the haptic rendering algorithms. This option specifies which sides of the haptic shapes to render haptically. Once the avatar falls inside the virtual object, it will become straightforward to get out of it by setting its value to *FRONT*, otherwise collision detection exists and the user will touch the inside surface which is not desirable. Additionally, by setting *useBoundTree* to *false*, a gain in the performance is obtained without affecting the perception to the user. For the 'Liver Medium' topology mesh without using bounding trees, an increase of 8 units in the fps rate was seen having as total 24 instead of 16.

Even when bound trees are used, the GeometryBoundTreeOptions help to decide which type of collision objects to use, either *AABB*, *OBB* or *SPHERE* as well as the maximum number of objects in the leafs of the tree. The three different types were useful and without any perceived difference.

Finally `DebugOptions` appear as disabled as these indeed affect performance. If enabled, a bound box is drawn in addition to the bounding tree structure used for collision detection. Also, once the virtual object is in touch, the closest triangles are highlighted with the purpose of showing which haptic shapes are involved during the haptic rendering.

Optimisation

Introduction

OpenMP was used as an approach to improve the frames per second rate in the Haptics Open Framework. A for loop directive was configured for data parallelism in the deformation modelling when the mechanical behaviour is used to fetch the new points position into H3D.

By using OpenMP, the test scenario was forced to be a debug environment in VS2005. However, as all the tests were carried out in a Release version compiled in VS2003 it was hard to do comparisons about the improvement by using parallelisation.

Tests and Results

Nonetheless, 'Liver Medium' was compared using the debug version. There was not significant increase in the frames per second rate shown in the framework. For this particular topology, an increase of 2 to 3 fps was achieved.

Force Fields for Biomechanical Modelling

Introduction

The purpose of this test is to compare the two available force field methods in SOFA and justify its usage in the Haptics Open Framework. For the test scenario, gravity was set to -9.8 in the y-axis with a total mass of 1kg. and `CGImplicit` solver with 25 iterations as maximum for the numerical integration. For `MeshSpring` force field stiffness was set to a value of 10,000 and damping to 15. For `TetrahedronFEM`, young modulus was set to 100 and poisson ratio to 0.4.

Tests and Results

`MeshSpring` force field is considered computationally efficient then real-time simulation is feasible. However as a discrete model it involves approximations of the true physics and small time steps are required. To avoid smalls time steps, a Conjugate Gradient approach is used as the solver (refer to Numerical Integration Solver Tests and Results for more information). Even when `TetrahedronFEM` is computationally more expensive, it can simulate the physical properties of an object more efficiently.

In Figure 30, the resulting frames per second in SOFA are shown given different levels of mesh accuracy. It is important to notice that the plotted fps were collected before the object reached equilibrium. As an example, for 'Liver Medium' mesh, `MeshSpring` is twice faster than its counterpart using FEM. `MeshSpring` has been chosen as a recommendation for the framework due to the constraints in processing resources and an approximation to the physics of a human organ.

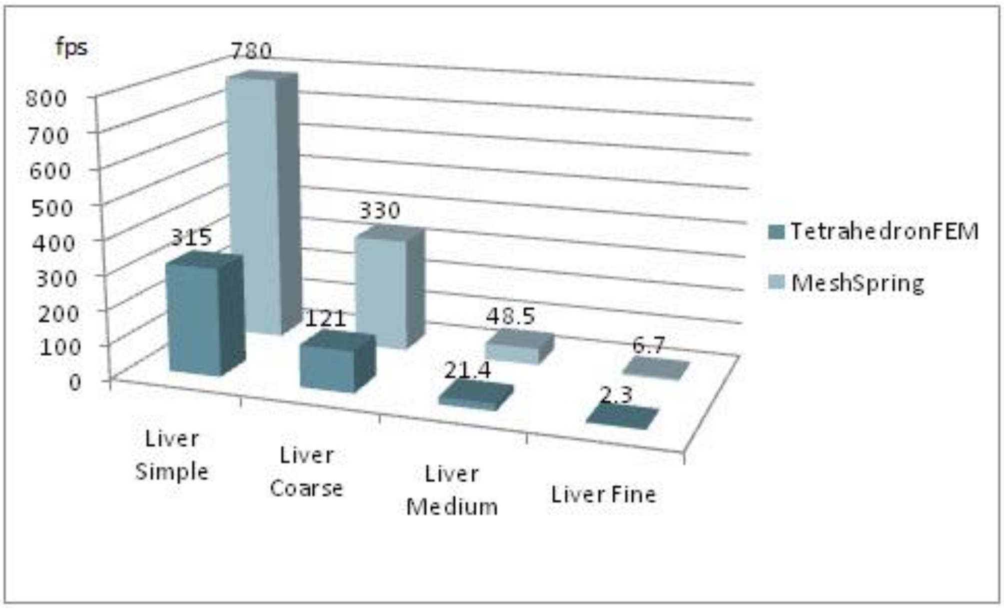


Figure 30 - SOFA Force Fields

Numerical Integration Solver

Introduction

Four different solvers are analysed using different mesh topology sizes in SOFA. The scene-graph has been set with gravity magnitude of -9.8 in the y-axis and using a `MeshSpringForceField` with stiffness and damping values of 10,000 and 15 respectively. Additionally, a total mass of 1Kg for all the nodes in the system is applied.

Tests and Results

Figure 31 shows a performance comparison of four different solvers available in SOFA: `Euler`, `RungeKutta4`, `StaticSover` and `CGImplicit`.

Euler solver was configured as symplectic for a semi-implicit Euler integration instead of the standard Euler method which collapsed for all mesh sizes. It is semi-implicit because it uses the next velocity to compute the next position. For the two lower resolution meshes, the numerical integration was unstable using small time step of 0.002. For the other two meshes, the nodes collapsed even when several time steps were tested suing symplectic Euler and RK4. Red-coloured label values show instability.

Additionally, it was observed that Static solver reached equilibrium so fast for low quality meshes. Even when it performs better than others, it was not adequate for the haptic deformable shapes.

On the other hand, `CGImplicit` with a time step of 0.02 showed efficient performance and is suitable for haptic deformation shapes.

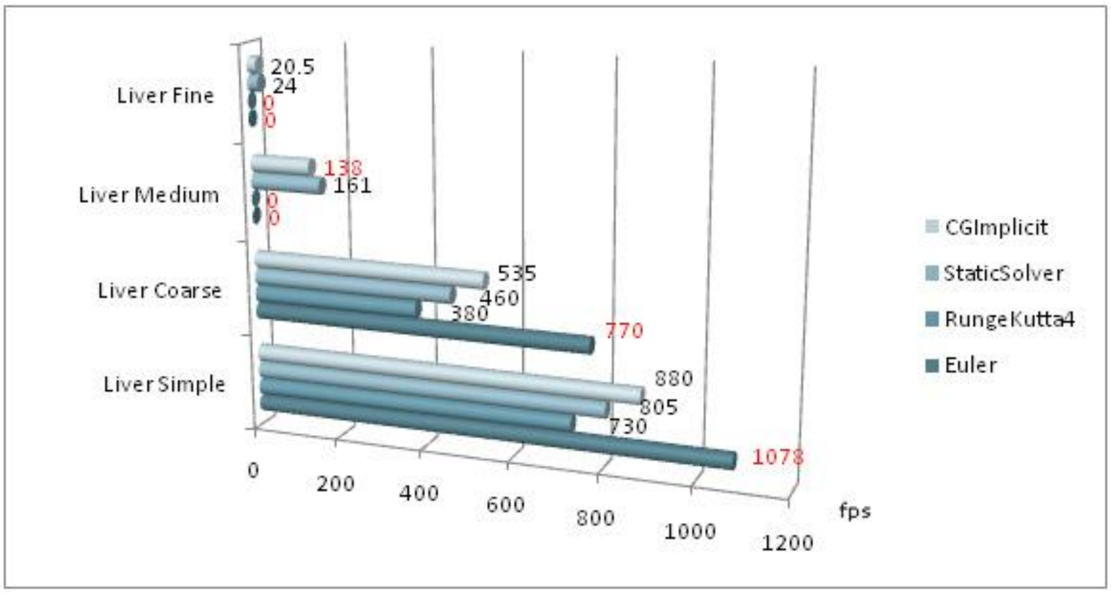


Figure 31 - SOFA Solvers Comparison

Even when CGImplicit performed better in the previous test, additional tests were carried out with different maximum iterations values using Static solver. Figure 32 shows the captured fps in SOFA. For higher resolution meshes and high number of iterations the simulation becomes unstable and slower.

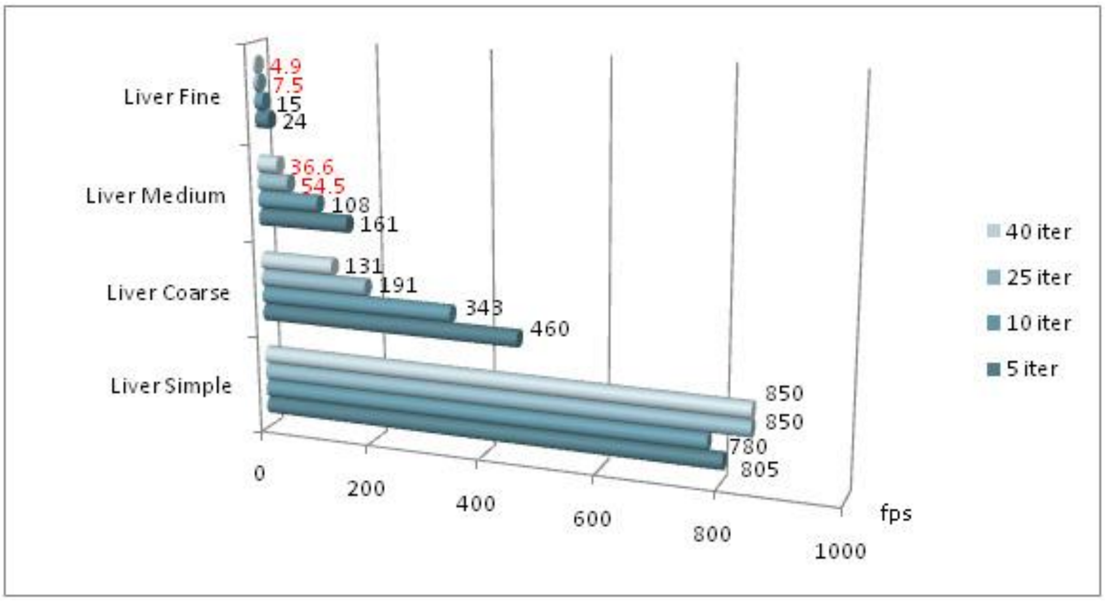


Figure 32 - Static Solver Performance

Finally, the same tests were performed in CGImplicit solver using different maximum number of iterations. Results are shown in Figure 33. The reason of not presenting instability when high resolution meshes and low number of iterations are used might be due to the slow deformation observed. For the particular example of 'Liver Medium' by changing the number of maximum number of iterations to seven it becomes stable in SOFA. However, once 'Liver Medium' is loaded into HoF, it was observed a better behaviour when setting the maximum number of iterations to a value around thirty. For 'Liver Simple' and 'Liver Coarse', one iteration as maximum is quite enough for the simulation in SOFA.

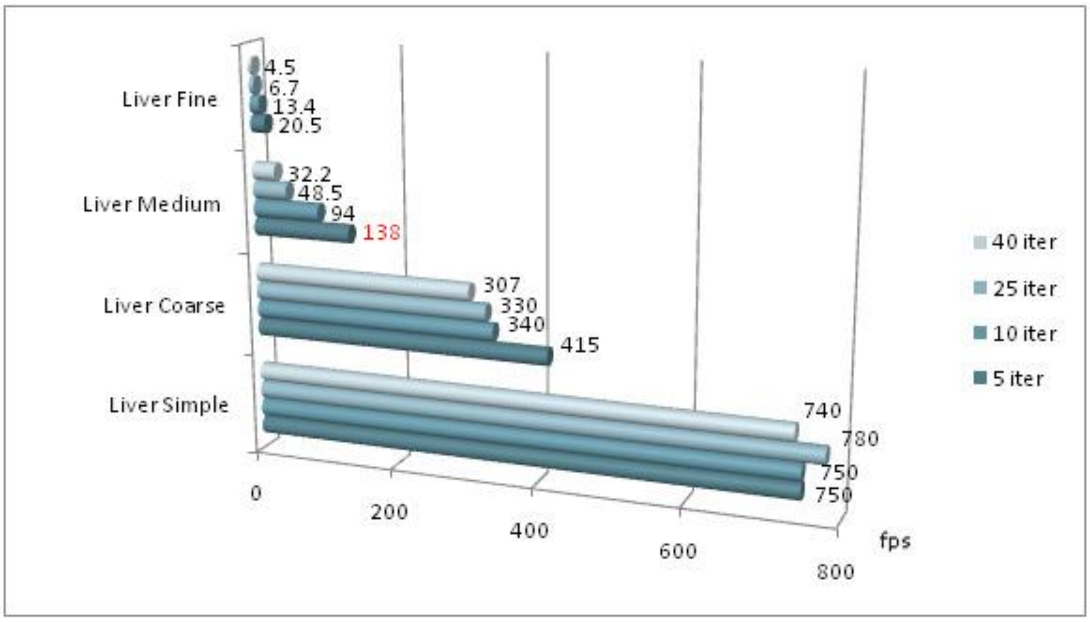


Figure 33 - CGImplicit Solver Performance (dt=0.02)

Haptics Open Framework Evaluation

Introduction

The main purpose of this section is to evaluate the performance of the Haptics Open Framework as an overall accordingly to all the decisions taken in its design and previous evaluation results.

Tests and Results

Figure 34 shows three frames per second rates achieved using the four different topology sizes for a human liver, that is, when the virtual object has not been touched by the avatar, when it is deforming due to a previous contact and while in contact. The best approach is by mixing deformation, haptics and graphics in real time and this is the particular case of 'Liver Medium'. For a higher resolution which is the case of 'Liver Fine' the simulation loop becomes slower and not real time so deformations are not efficiently perceived, even though the collision detection does.

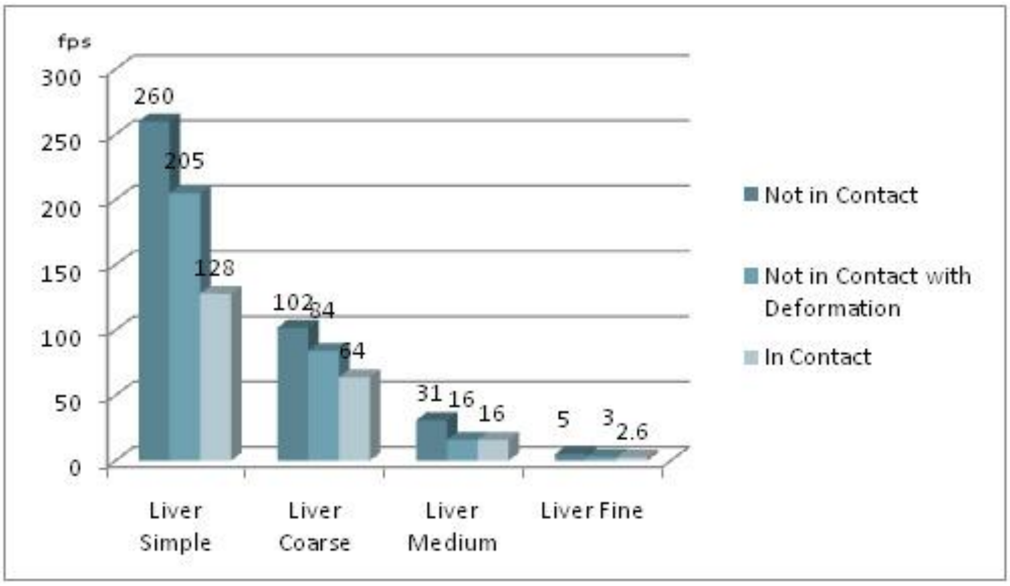


Figure 34 - HoF Performance

Figure 35 shows a snapshot of how these different topology sizes are deformed when in touch with their respective fps rate.

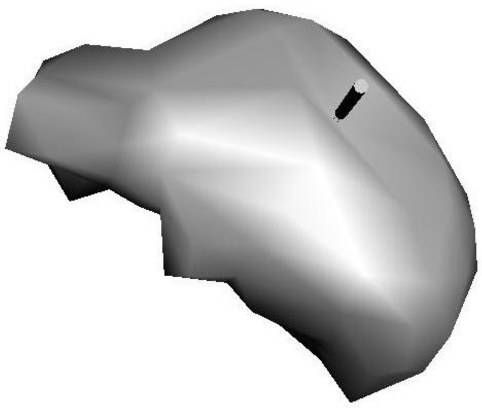
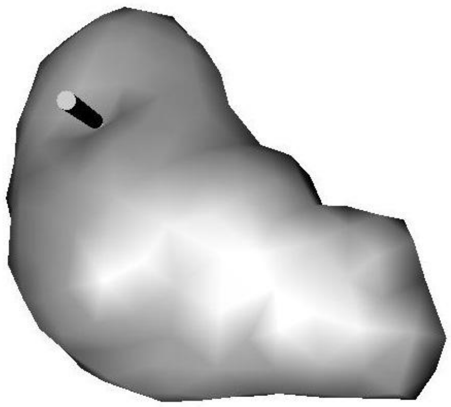
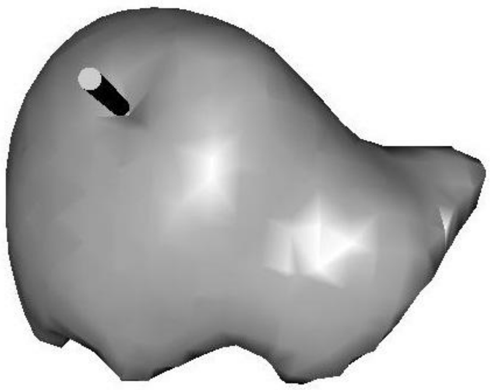
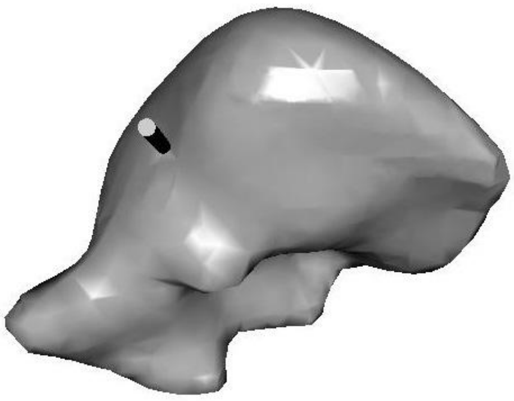
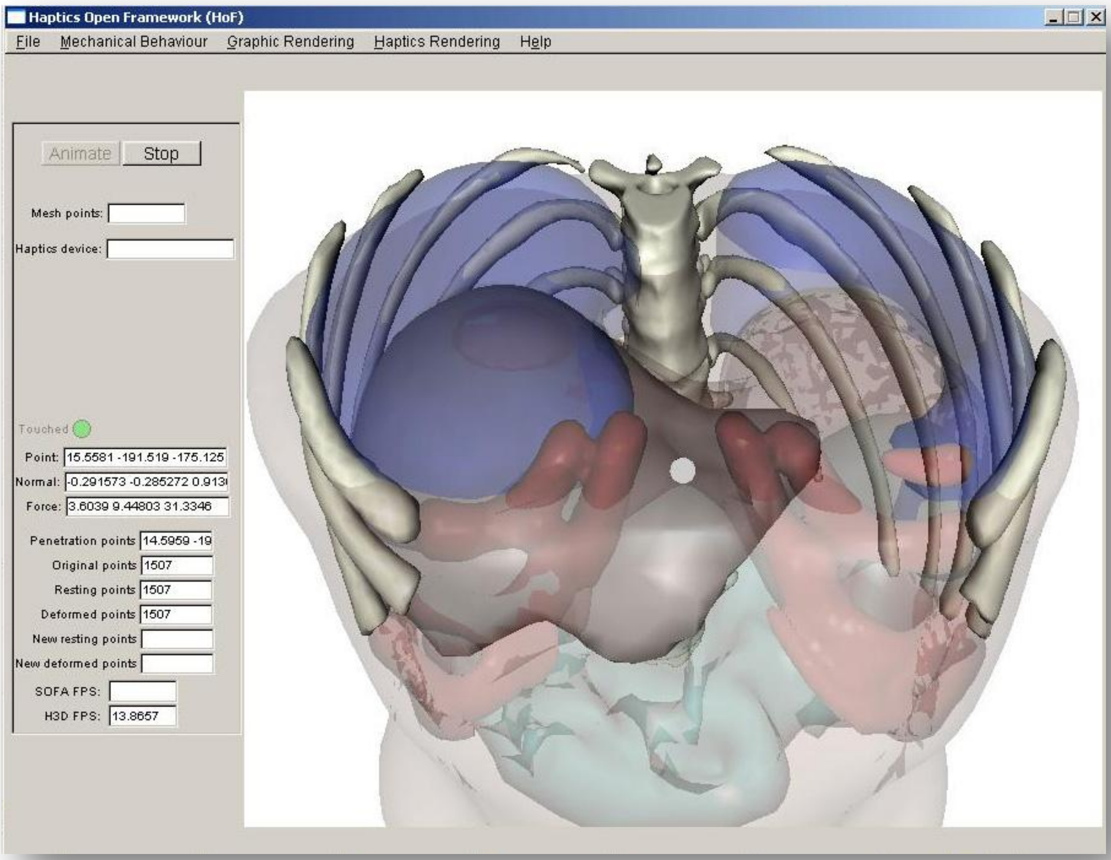
Liver Simple (128fps)	Liver Coarse (64fps)
	
	
Liver Medium (16fps)	Liver Fine (2.6 fps)

Figure 35 - Haptic Deformable Shapes Examples

Finally, a framework screenshot is shown in Figure 36 using a touched ‘Liver Medium’ within a 3D environment using several organs and bones which are closed to the liver to achieve a more realistic simulation scenario for a user. All organs and bones different than the liver are not touchable and transparency values in their appearance model have been set to allow the user to view the liver.



Mesh points:
Haptics device:
Point: 15.5581 -191.519 -175.125
Normal: -0.291573 -0.285272 0.913
Force: 3.6039 9.44803 31.3346
Penetration points 14.5959 -19
Original points 1507
Resting points 1507
Deformed points 1507
New resting points
New deformed points
SOFA FPS:
H3D FPS: 13.8657

Figure 36 - HoF Liver Deformation

Needle Insertion

Introduction

Needle insertion modelling has been described in the implementation chapter. In this section some tests and results pretend to give a feedback on what has been implemented and if the resulting forces and perception to the user is efficient enough for this kind of procedures.

The topics that will be tested include the avatar graphical representation of the needle, computing penetration depth, setting a threshold value for the capsule in the force effect node and test if forces are generated based on it. Additionally deformation will be tested by sending the computed forces to SOFA. The perception of these procedures is quite important and represents the main outcome of these tests.

Tests and results

Even when the graphical representation of the avatar for the stylus was quite simple it indeed represents a needle and a syringe, then the perception to the user becomes more realistic. However, this is just a syringe representation as the needle is merely specified for its tip which longitude is only 94mm

Also, even when the computation of the penetration depth based on touch point and penetration point is realistic, it was not clearly observed when the needle was traversing the organ without setting the transparency value. Because of this, a small sphere is drawn in the tip of the needle and a red point is drawn at the touch point to emphasize the insertion.

It is important to notice that deformation is done in the closest vertex not in the touch point and as forces are small the only way to notice a deformation is by changing the stiffness of the virtual object. Nonetheless, forces are computed accordingly to Maurin et al [2004] paper and the negative of the touch normal is used to set the force which will be sent to SOFA for deformation modelling.

Needle extraction behaves exactly the same as the insertion phase, that is, forces are computing merely by the penetration depth.

In Figure 37, two examples of needle insertion are shown by using a cube geometry. No deformation is applied in aims to show the appearance of the procedure. As it could be observed, transparency has been set in the appearance model in order to have a better perception of the needle insertion. The sphere drawn in the needle's tip helps to identify the penetration depth.

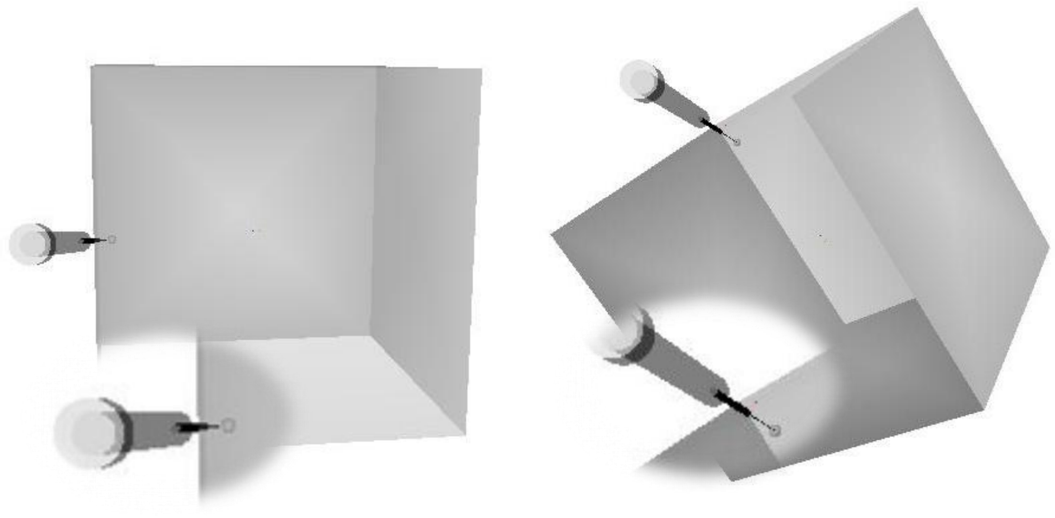


Figure 37 - Needle insertion in a transparent cube (no deformation)

Introducing deformation in this needle insertion experiment became quite straightforward which is an advantage to the proposed Haptic Open Framework. Figure 38 shows a couple of screenshots of the 'Liver Medium' topology using needle insertion and deformation.

It can be seen that the sphere drawn in the tip of the needle does not appear in the surface representing that the needle has penetrated the organ. At the same time, both examples show deformation in the contact region. However, stiffness value was decreased from 5,000 to 1,000 in order to be able to show deformation in a clearer form.

For this example, a value of 0.02m was used for the capsule, that is, the distance from the liver surface to the liver capsule. This value was chosen as an approximation based on the graph presented in Figure 21. As this distance is small, it becomes complicated to position the needle's tip in a region between the surface and the capsule to identify the forces during the first phase of Fung model. Nonetheless, even when forces were slightly felt in the haptics device, the exponential increment in the forces as shown in Figure 21 was perceived.

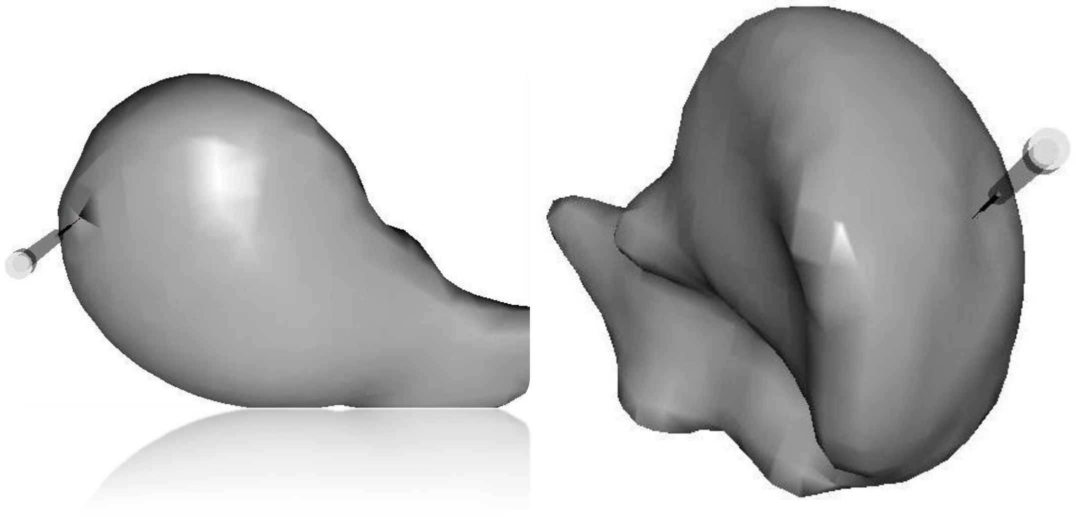


Figure 38 - Needle Insertion in a liver (with deformation)

Due to the limited workspace of the Falcon haptic device and the small forces exerted into the device, it was not possible to insert the needle further after the initial contact with the capsule.

Conclusion and Future Work

Several research efforts in a myriad of different areas are involved in aims to present an efficient alternative to surgical training using simulation in real time. By combining recent available methods in haptic rendering, modelling of deformable objects and using open source frameworks, nowadays surgery training is becoming a fact.

Haptic rendering provides immeasurable benefits for a realistic experience, especially in medical training. Consumer-based products such as NOVINT®'s Falcon haptic device make efforts to gather even better as a result of cost and availability. Collision detection and force response play an important role due to the processing demand in high-frequency servo rates.

Although most of the physically-based models of deforming objects (continuum mechanical-based approaches) introduce the best approximation for the dynamic behaviour of soft tissue, they still present challenges when dealing with limited computational resources. Furthermore, the choice of such algorithms, together with the data structures selected for representing volume, have a direct impact on typical tasks performed in medical training, such as cutting and grasping as this tasks imply topology changes.

Nonetheless, recent open source software frameworks available both in haptic rendering and simulation engines grant benefits never encountered before, owing to the modularity and flexibility for using available algorithms and deploying new ones. Furthermore, these frameworks provide mechanisms to do significant changes in a relative short period of time, allowing for further outcome comparison and analysis.

Overall, even though accuracy and efficiency can be found in several approaches, trade-offs are still of paramount concern to achieve realistic scenarios in immersive and interactive environments.

Using both volumetric and surface information is important for modelling touchable deformable shapes and combining them together in a real-time application is challenging. Thereafter, this is an important key element of the Haptics Open Framework.

The integration of SOFA and H3D in aims for a haptics-enabled framework for deformable shapes is quite important due to the flexibility of extending these libraries in order to implement and test recent algorithms and to be device independent as well as a method for creating virtual scenarios.

Even when the graphical user interface integration was complicated in the first part of the project, it could be managed satisfactory. However, as there are many values and properties that can be set in both SOFA and H3D libraries, it becomes a time-consuming activity to create their corresponding widgets, especially for handling both scene-graphs.

The proposed framework provides several benefits for quick prototyping of deformable objects and a platform to test different algorithms for modelling haptics and deformation. Also, academically this framework might be useful for teaching purposes in the field of haptics programming and deformation modelling which indeed extends its use.

A particular example of prototyping was implemented into the framework for needle insertion procedures such as biopsies. Even though the implementation can be improved, it was quick implemented using the framework and deformation was straightforward to handle.

Future work will include a better appearance modelling which is of extreme importance to present realistic scenarios. Also, nonlinearity deformation needs to be included due to the nature of soft tissue. The use of massive parallelisation in computing might enhance the performance and allow for more techniques to adapt such as mesh refinement.

Additionally, the graphical user interface might support the X3D scene graph in the framework. By modelling grasping and cutting, even more medical procedures can be simulated and different tools might be integrated.

An important part of training is the measure of performance thus statistics will be of paramount importance for the practitioners in a virtual environment.

References

ALLARD, J., COTIN, S., FAURE, F., BENSOUSSAN, P.-J., POYER, F., DURIEZ, C., DELINGETTE, H., AND GRISONI, L.. 2007. “SOFA – an Open Source Framework for Medical Simulation”. CIMIT Sim Group and INRIA. 1-6.

AMRANI, M., JAILLET, F., AND SHARIAT, B. 2000. “Deformable Objects Modelling and Animation: Application to Organs’ Interactions Simulation”. University of Lyon. *Journal for Geometry and Graphics, Volume 4, 2000*, 181-188.

BARAFF, DAVID AND WITKIN, ANDREW. 1998. “Large Steps in Cloth Simulation”. Robotics Institute, Carnegie Mellon University. 1-12.

BARGTEIL, ADAM W., WOJTAN, CHRIS, HODGINS, JESSICA K., AND TURK, GREG. 2007. “A Finite Element Method for Animating Large Viscoplastic Flow”. Carnegie Mellon and Georgia Institute of Technology. *SIGGRAPH 2007*, 1-8.

BRO-NIELSEN, MORTEN. 1996. “Surgery Simulation using Fast Finite Elements”. Technical University of Denmark. 1-6.

CHOI,KUP-SZE, SUN, HANQIU, HENG, PHENG-ANN, AND ZOU, JUN. 2004. “Deformable Simulation Using Force Propagation Model With Finite Element Optimization”. Chinese University of Hong Kong. *Computer and Graphics*, 559-568.

CHOI, SEUNGMOON AND TAN, HONG Z. 2004. “Toward Realistic Haptic Rendering of Surface Textures”. Purdue University. *IEEE Computer Graphics and Applications*, 40-47.

CONTI F, BARBAGLI F, MORRIS D, SEWELL C. 2005. “CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes”. IEEE World Haptics.

DEHGHAN, EHSAN, WEN, XU, ZAHIRI-AZAR, REZA, MARCHAL, MAUD, AND SALCUDEAN, SEPTIMIU E. 2007. “Modeling of Needle-Tissue Interaction Using Ultrasound-Based Motion Estimation”. University of British Columbia and TIMC-GMCAO Laboratory. *MICCAI 2007*, 709-716.

FARACI, ALESSANDRO. 2005. “A Multi-resolution Nonlinear Finite Element Approach to Real-time Simulation of Soft Tissue Deformation with Haptic Feedback”. Doctoral Dissertation, Department of Surgical Oncology and Technology, Imperial College London.

FAURE, FRANCOIS. 2007. “Introduction to SOFA”.Grenoble Universités, 1-13.

FAURE, FRANCOIS, ALLARD, J., COTING, S., NEUMANN, P., BENSOUSSAN, P-J., DURIEZ, C., DELINGETTE, H. AND GRISONI, L. 2007. “SOFA: A Modular Yet Efficient Simulation Framework”. *Surgetica 2007*, 1-7.

FLTK. “Fast Light Tool Kit 1.1.7 Programming Manual”. http://www.fltk.org

FREEGLUT. “The Open-Source OpenGL Utility Toolkit Freeglut API”. http://freeglut.sourceforge.net

GAUZAINE, C. AND REMACLE, J.-F. 2007 “Gmsh Reference Manual”. http://geuz.org/gmsh

GEORGII, JOACHIM AND WESTERMANN, RÜDIGER. 2005. “Mass-Spring Systems on the GPU”. Technische Universität München. 1-10.

GIBSON, SARAH F. F. 1996. “3D ChainMail: A Fast Algorithm for Deforming Volumetric Objects”. Mitsubishi Electric Research Laboratories. 1-7.

GLEW. “GLEW: The OpenGL Extension Wrangler Library”. http://glew.sourceforge.net

GLUT. "The OpenGL Utility Toolkit (GLUT) Programming Interface (API version 3)"

GNU. "The GNU Operating System". http://www.gnu.org

GOKSEL, ORCUN, SALCUDEAN, SEPTIMIU E. AND DIMAIO, SIMON P. 2006. "3D Simulation of Needle-Tissue Interaction With Application to Prostate Brachytherapy". University of British Columbia, and Brigham and Women's Hospital. *Computer Aided Surgery*, 279-288.

H3D OPEN SOURCE HAPTICS. "H3D API Manual (for version 2.0)". http://h3dapi.org

HALE, KELLY S. AND STANNEY, KAY M. 2004. "Deriving Haptic Design Guidelines from Human Physiological, Psychophysical, and Neurological Foundations". University of Central Florida. *IEEE Computer Graphics and Applications*, 33-39.

HAYWARD, VINCENT AND MAHVASH MOHSEN. 2004. "High-Fidelity Haptic Synthesis of Contact with Deformable Bodies". McGill University and Real Contact Inc. *IEEE Computer Graphics and Applications*, 48-55.

HOLBREY, RICHARD PAUL. 2004. "Virtual Suturing for Training in Vascular Surgery". Doctoral Dissertation, School of Computing, The University of Leeds.

ISBMS. 2008. "International Symposium on Computational Models for Biomedical Simulation". Imperial College London. July 7th and 8th, 2008.

KIM, LAEHYUN, SUKHATME, GAURAV S. AND DESBRUN MATHIEU. 2004. "A Haptic-Rendering Technique Based on Hybrid Surface Representation". Korea Institute of Science and Technology and University of Southern California. *IEEE Computer Graphics and Applications*, 66-75.

KYTHE, PREM K. 1995. "An Introduction to Boundary Element Methods".

LIBXML2. "The XML C Parser and Toolkit for Gnome". http://xmlsoft.org

MAURIN, B., BAYLE, Z., GANGLOFF, DE MATHELIN, GANGI, S., AND FORGIONE. 2004. "In Vivo Study of Forces During Needle Insertions". 1-8.

MCLAUGHLIN, MARGARET L. 2005. "Touch in virtual environments: haptics and the design of interactive systems"

MEIER, U., LÓPEZ, O., MONSERRAT, C., JUAN, M.C., ALCAÑIZ, M. 2005. "Real-time Deformable Models for Surgery Simulation: a Survey". Universidad Politécnica de Valencia. *Computer Methods and Programs in Biomedicine 2005*, 183-197.

MONSERRAT, C., MEIER, U., JUAN, MC., ALCAÑIZ, M., KNOLL, C., GRAU, V., CHINESTA, F., AND DUVAL, C. 2001. "A New Approach for the Real-Time Simulation of Tissue Deformations". Universidad Politécnica de Valencia and Conservatoire National des Arts et Metiers Paris, 1-8.

MOORE, PATRICIA AND MOLLOY, DEREK. 2007. "A Survey of Computer-Based Deformable Models". Machine Vision and Image Processing Conference. Dublin City University, 1-10.

NEDEL, LUCIANA PORCHER, THALMANN, DANIEL. 1998. "Real Time Muscle Deformations Using Mass-Spring Systems". Swiss Federal Institute of Technology, 1-11.

NEWMAT. "Robert Davies' NewMAT library". http://www.robertnz.net

NIENHUYS, AND FRANK VAN DER STAPPEN. 2003. "Interactive Needle Insertions in 3D Nonlinear Material". Utrecht University, 1-7.

NOVINT. "NOVINT Pioneering Interactive 3D Touch Products". http://www.novint.com

OGRE. "OGRE 3D: Open Source Graphics Engine". http://www.ogre3d.org

OPENGL. "OpenGL – The Industry Standard for High Performance Graphics". http://opengl.org

OPENGL. "The OpenGL Graphics System Utility Library (version 1.3)".

OPENMP. "The OpenMP API Specification for Parallel Programming Tutorial ". http://openmp.org/wp

PALOC, CÉLINE, FARACI, ALESSANDRO, AND BELLO, FERNANDO. 2006. "Online Remeshing for Soft Tissue Simulation in Surgical Training". VICOMTech and Imperial College London. *IEEE Computer Graphics and Applications*, 24-34.

QT4WIN GUI. "Trolltech Qt 4.2 Reference Documentation". http://doc.trolltech.com/4.2/index.html

RUSPINI, D., KOLAROV, K. AND KHATIB, O. 1997. "The Haptic Display of Complex Graphical Environments". *Computer Graphics Annual Conference Series*, 345-352.

SALISBURY, KENNETH AND LIN, MING. 2004. "Haptic Rendering – Beyond Visual Computing". *IEEE Computer Graphics and Applications*, 22-23.

SALISBURY, KENNETH, CONTI, FRANCOIS AND BARBAGLI, FEDERICO. 2004. "Haptic Rendering: Introductory Concepts". Stanford University and University of Siena. *IEEE Computer Graphics and Applications*, 24-32.

SALISBURY, KENNETH AND TARR, C. 1997. "Haptic Rendering of Surfaces Defined by Implicit Functions". *ASME Dynamic Systems and Control Division vol. 61*, 61-67.

SENSABLE. "SensAble Technologies". http://sensable.com

SENSGRAPHICS. "SenseGraphics H3D API". http://www.sensegraphics.com

SMART, J., ROEBLING, R., ZEITLIN, V., DUNN, R., ET AL. November, 2007. "wxWidgets 2.8.7: A portable C++ and Python GUI toolkit". http://www.wxwidgets.org

SOFA. "SOFA :: Simulation Open Framework Architecture". http://www.sofa-framework.org

TAYLOR, ZEIKE A., CHENG, MARIO, AND OURSELIN, SÉBASTIEN. 2007. "Real-Time Nonlinear Finite Element Analysis for Surgical Simulation Using Graphics Processing Units". CSIRO ICT Centre and University College London. *MICCAI 2007*, 701-708.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. "Elastically Deformable Models". Schlumberger Palo Alto Research and California Institute of Technology. *Computer Graphics 1987*, 205-214.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. "Collision Detection for Deformable Objects". University of Freiburg, ETH Zurich, Tübingen University, University of Geneva, INRIA, Bonn University, and Fraunhofer Institute for Computer Graphics. *The Eurographics Association and Blackwell Publishing 2005*, 1-21.

VIDHOLM, ERICK. 2008. "Visualization and Haptics for Interactive Medical Image Analysis". Doctoral Dissertation, Faculty of Science and Technology, Uppsala Universitet.

WITKIN, ANDREW. 1997. "Physically Based Modeling: Principles and Practice. Particle System Dynamics". Carnegie Mellon University. 1-12.

WU, XUNLEI, DOWNES, MICHAEL S., GOKTEKIN, TOLGA, AND TENDICK, FRANK. 2001. “Adaptive Nonlinear Finite Elements for Deformable Body Simulation Using Dynamic Progressive Meshes”. University of California. *EUROGRAPHICS 2001*, 1-10.

X3D. “X3D (Extensible 3D): Part 1: Architecture and Base Component Specification”. http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1

ZAMANI, K., TOMOYUKI, M., AND YAMAKAWA, H. 2007. “A Simulation Method for 3-Dimension Needle Invasive Medical Treatments of Cancer Infected Liver”. Biomedical Engineering Proceedings of the Fifth IASTED International Conference 2007, 43-49.

ZHU, YANONG, MAGEE, DEREK, RATNALINGAM, RISH, AND KESSEL, DAVID. 2007. “A Training System for Ultrasound-Guided Needle Insertion Procedures”. University of Leeds, Mid Yorkshire Hospitals and Leeds Teaching Hospitals. *MICCAI 2007*, 566-574.

ZIENKIEWICZ, O. C. AND TAYLOR, R. L. 2000. “The Finite Element Method”.

ZILLES, C.B. AND SALISBURY, J.K. 1995. “A Constraint-based God-object Method For Haptic Display”. MIT, 1-6.

Appendix

XML Project Settings

An example of a framework project file for “Liver Coarse” is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<hof>
  <sofa>
    <sg>
      <Node name="root" dt="0.02" showBehaviorModels="0"
               showCollisionModels="0" showMappings="0"
               showForceFields="0" gravity="0 0 0">
        <Object type="CGImplicit" iterations="5"/>
        <Object type="MeshTopology" name="mesh"
               filename=" livercoarse.msh" />
        <Object type="MechanicalObject" name="mechanics"
               template="Vec3d" />
        <Object type="FixedConstraint" name="fixed"
               indices="11 86 87 124 128"/>
        <Object type="ExternalForceField" name="extforce"
               force="0 0 0" />
        <Object type="MeshSpringForceField" name="spring"
               stiffness="1000" damping="5" />
        <Object type="UniformMass" name="mass" totalmass="1000"/>
      </Node>
    </sg>
  </sofa>
  <h3d>
    <graphics fullscreen="no" mirrored="no" renderMode="MONO">
      <viewpoint filename="ViewpointH3D.x3d "/>
    </graphics>
    <haptics showTime="no" showLevel="no" outputLevel="3" stream="cerr">
      <device filename="DeviceFalcon.x3d"/>
      <stylus filename="StylusH3D.x3d"/>
      <geometry filename="livercoarse.x3d"/>
    </haptics>
  </h3d>
</hof>
```

X3D Scene-Graph Example

An example of an X3D scene-graph is shown below.

```
<Group>
  <FitToBoxTransform boxCenter="0 0 0" boxSize="0.4 0.4 0.4">
    <Transform>
      <DeformableShape>
        <Motion DEF="box2"/>
        <Appearance>
          <SmoothSurface />
          <Material
            ambientIntensity='1.0'
            diffuseColor='0.36 0.11 0.04'
            emissiveColor='0 0 0'
            shininess='0.2'
            specularColor='0.70 0.70 0.70'
            transparency='0'
            />
        </Appearance>
        <IndexedTriangleSet  DEF="box"   solid='false'
            normalPerVertex='true' containerField='geometry'
            index='
                  ...
                  ' >
          <Coordinate point='
                  ...
                  '/>
        </IndexedTriangleSet>
      </DeformableShape>
    </Transform>
  </FitToBoxTransform>
</Group>
```

H3D Source Code Modifications

GLUTWindow

Firstly, include the corresponding libraries for FLTK and replace Freeeglut GLUT's implementation by FLTK. Also, all the GLUT callback functions for the keyboard and mouse interaction will be enabled.

```
////////////////////////////////////
/ GLUTWindow.h Modifications
////////////////////////////////////
#include <FL/Fl_Window.h>

#ifdef FLTK
#include <FL/glut.h>
#endif
#ifdef FREEGLUT
#include <GL/glut.h>
#endif

class H3DAPI_API GLUTWindow : public H3DWindowNode,
      public Fl_Window { … }
```

Thereafter, `H3D::GLUTWindow` implementation adds the library dependencies dynamically. When mixing FLTK and GLUT code we don't need to call `glutInit()` function, then it is commented in `GLUTWindow::initGLUT()` function. The constructor of `Fl_Window` is called accordingly.

```
////////////////////////////////////
/ GLUTWindow.cpp Modifications
////////////////////////////////////
#ifdef FLTK
#include <FL/glut.h>
#include <GL/glu.h>        // important for FLTK
#define  GLUT_INIT_STATE   0x007C
#endif
#ifdef FREEGLUT
#include <GL/glut.h>
#endif

#ifdef FLTK
#if defined(_MSC_VER) || defined(__BORLANDC__)
#pragma comment( lib, "comctl32.lib")
#pragma comment( lib, "fltkdlld.lib" )
#pragma comment( lib, "glu32.lib")
```

```
#pragma comment( lib, "opengl32.lib")
#endif
#endif

void GLUTWindow::initGLUT() {
  if ( !GLUT_init ) {
    // don't call glutInit when mixing FLTK and GLUT
    //char *argv[1] = { "H3DLoad" };
    //int argc = 1;
    //glutInit(&argc, argv);
    GLUT_init = true;
  }
}

GLUTWindow::GLUTWindow( … ) : H3DWindowNode( … ),
      Fl_Window(0,0,0,0, "GLUTWindow") {…}
```

The most important changes are done in the `GLUTWindow::initWindow()` function. As the GUI will be the parent window, it needs to be shown first. The next step is to show the child window and handle it as another widget. In H3D, when a Scene is created, the GLUTWindow is pushed back in a vector and this is the time when the window initialization takes place. The `GLUTWindow::initWindow()` code changes are as follows:

```
show(*intargc, intargv);          //fltk
begin();                          //fltk
glutInitDisplayMode( mode );
glutInitWindowSize( w(), h());
glutInitWindowPosition(0,0);            //fltk
window_id = glutCreateWindow( "H3D" );
end();                            //fltk
resizable(glut_window);                 //fltk
```

Scene, X3DKeyDeviceSensorNode, DynamicTransform, and SpaceballSensor

Similar to GLUTWindow, all these classes need to replace Freeglut's GLUT implementation by FLTK implementation as follows:

```
#ifdef FLTK
#include <FL/glut.h>
#endif
#ifdef FREEGLUT
#include <GL/glut.h>
#endif.
```

X3D Needle Representation

This section includes the X3D representation for the stylus used in a needle insertion procedure. It is divided in different sections or shapes which constitute a syringe and a needle. These shapes appear form the tip towards the syringe and are commented accordingly. As it could be observed, Transform nodes are used to set translation values for each shape and just the z-axis, which is towards the user, is modified. Also, rotation values are set to $\pi/2$ in order to have the syringe facing towards the virtual object.

```
<Group>
  <Shape>
    <!--needle's tip-->
    <Appearance>
      <Material transparency="0.5"/>
    </Appearance>
```

```
      <Sphere radius="0.0025"/>
    </Shape>
    <Transform translation="0 0 0.025" rotation="1 0 0 1.570796" >
      <Shape>
        <!--needle tip diameter-->
        <Appearance>
          <Material/>
        </Appearance>
        <Cylinder radius="0.00025" height="0.047"/>
      </Shape>
    </Transform>
    <Transform translation="0 0 0.075" rotation="1 0 0 1.570796" >
      <Shape>
        <!--second half of the needle, different diameter-->
        <Appearance>
          <Material/>
        </Appearance>
        <Cylinder radius="0.00125" height="0.047"/>
      </Shape>
    </Transform>
    <Transform translation="0 0 0.15" rotation="1 0 0 1.570796" >
      <Shape>
        <!--syringe tube base-->
        <Appearance>
          <Material transparency="0.5"/>
        </Appearance>
        <Cylinder radius="0.005" height="0.1"/>
      </Shape>
    </Transform>
    <Transform translation="0 0 0.2" rotation="1 0 0 1.570796" >
      <Shape>
        <!--syringe handle-->
        <Appearance>
        <Material transparency="0.5"/>
        </Appearance>
        <Cylinder radius="0.008" height="0.01"/>
      </Shape>
    </Transform>
  </Group>
```