Terje Sanderud Haaland

Virtual Reality Techniques in the Domain of Surgical Simulators

Trondheim, 2006

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

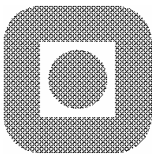
Master's thesis
Programme of study: Master of Science in Computer Science

Supervisor: Torbjørn Hallgren, IDI
Co-Supervisor(s): Odd Erik Gundersen, IDI

NTNU
Innovation and Creativity

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK



MASTEROPPGAVE

Kandidatens navn: Terje Sanderud Haaland

Fag: Datateknikk

Oppgavens tittel (norsk): Teknikker for virtuell virkelighet med hensyn på kirurgisimulatorer

Oppgavens tittel (engelsk): Virtual reality techniques in the domain of Surgical simulators

Oppgavens tekst:

Over the last decades, modern technology continously has offered new applications within medicine. The new technology have contributed both in training of medical personnel and in operative environments.Computer graphics, Haptic interface and Physical modeling are all essential components of a surgical simulator.

This thises will study the priciples and concepts regarding surgical simulators. The main goal will be to develop a prototype of a surgical simulater where the main focus should be on ceeping the design conceptually clear and easy to understand. The prototype should be easy to expand and change.

Oppgaven gitt:	29. september 2005
Besvarelsen leveres innen:	6. april 2006
Besvarelsen levert:	6. april 2006
Utført ved:	Institutt for datateknikk og informasjonsvitenskap.
Veileder:	Odd Erik Gundersen

Trondheim, 6.april 2006

Torbjørn Hallgren
Faglærer

Abstract

Virtual reality based surgical simulators offer an elegant approach to enhancing traditional training in surgery. For interactive surgery simulation to be useful, however, there are several requirements needed to be fulfilled. A good visualisation is needed. The physical behavior of an organ must be modeled realistically. Finally there is a need of a device capable of force feedback to realise the possibility of "feeling" a virtual object.

In this thesis a basic prototype was developed to demonstrate all necessary concepts needed in a surgical simulator. The study was aimed at finding a suitable architecture and design for development of a surgical simulation application which is conceptually clear and easy to comprehend. Moreover, it was considered important that the prototype can provide a good basis for further experimentation. The main focus was on finding a satisfactory method that demonstrates the main concepts, while keeping the complexity as low as possible.

In the developed prototype, the visual impression of 3D is present, the haptic feedback works satisfactory, and the physical modelling proved to be feasible for simulating a virtual object. The object oriented design resulted in a compact and clear application, where changes in the implementation can be applied locally without unwanted implications elsewhere in the code.

Due to these qualities, implementing multi resolution and cutting was an easy task. Only minor changes to limited parts of the application was needed. This shows its suitability as a starting point for future experimenting and demonstration of consepts, in the field of surgical simulation.

Preface

This study is carried out as a part of Terje Sanderud Haaland's MSc studies at the Norwegian Uneversity of Technology and science, Faculty of Information Technology, Mathematics and electrical engeneering, Department of Computer and Information Science. The study is the result of a master thesis in the field og Algorithm constructions and Visualisation.

I would like to thank my supervisor Odd Erik Gundresen for his guidance throughout the development, and for his excellent advice on the writing the thesis.

I would also like to thank Silje Kufaas Tellefsen for invaluable encouragment and support reading through the study. I would like to thank Geir Kjetil Ferkingstad Sandve for many hours of helpful guidance and rich discussions. Finally I would like to thank Håvard Sjøvold for help and support during the main development periode for the prototype.

Trondheim 6. April 2006

Terje Sanderud Haaland

Contents

Chapter 1

Introduction

In this chapter we present the subject of this thesis. In section 1.1 we give a short introduction to the topic Surgical Simulators. In section 1.2 we state the objective and the requirements of this thesis. At the end we present the structure of this paper.

1.1 Surgical simulators

Modern computer science continuously offers new applications within medicine. Both within the training of medical personnel as well as during surgical intervention the new technology has developed and improved the routines.

Based on the scientific accomplishments of flight simulators and 3-D graphics visualisation, the virtual reality for surgical simulation and medical training began in the late 1980's [SJ98]. At this time, the systems were neither especially realistic, nor interactive. They were mainly used for examining the human anatomy and foreseeing consequences of medical interventions. Only some years later there was an intense progress in this area, where access to real human data was obtained. The "Visible Human Project" was a project producing one of the first data set containing anatomical data from a whole human being.

Moreover, there was also a vast development in the area of haptic devices, a device that let the user move a tool in three dimensions which aigain controls a

virtual tool on the computer screen. The haptic device is capable of producing physical force in all three degrees of freedom, providing the feeling of of a virtual object. Together with visual feedback, haptic devices are powerful tools in a wide range of applications, including medicine.

Training of surgeons is a major motivation for developing surgical simulators. Education of medical personnel today has a great potential for improvement. Currently most of the training is done on dead bodies and animals. The differences between dead and living tissue taken into consideration, this is not very satisfactory. On top of this the human and animal anatomy is not alike. Due to these weaknesses, there is a great need for students to operate on live persons under the supervision of experienced surgeons. But the risks and costs this involve, make it desirable to reduce the need of performing operations under supervision as much as possible. Although surgical simulators never can fully compensate for the training on live humans, they are expected to become as important for training of medical personnel as flight simulators are for pilots today [BTB+04].

Bro-Nielsen [BN98] point out three essential components that a surgical simulator needs:

Computer graphics: Graphics is needed to render realistic views of the virtual surgery scene and provide the surgeon with a visual illusion of reality.

Haptic interface: Haptic interfaces can provide the users with a physical sensation of touching and sensing objects in the virtual scene using force-feedback techniques. Use of surgical simulations requires that the operator "feel" the resistance of the object that is being virtually manipulated. That is, the action of the user results in a reaction, reflecting the mechanical characteristics of the virtual object.

Physical modeling: Physical models provide the surgeon with a behavioural illusion of reality. By modeling the visco-elastic deformation of human skin, like the fluid flow of blood from a wound, the models ensure that the virtual scene reflects the behaviour of the physical reality.

Implementing a realistic, full functioning, surgical simulator that fulfills all of these requirements, would be far to extensive for a Master thesis.

1.2 Objective

The objective of this thesis is to design and implement a basic prototype of a surgical simulator. This prototype should be suitable for reuse and experimenting by future students and researchers.To meet these demands, the following requirements are put forward:

- Virtual objects should be visualised in 3D
- A Haptic module making it possible to touch and feel virtual objects in the scene
- Physical simulation of a virtual object. That is, the object must respond when touched by a virtual tool controlled by the haptic device. E.g. like an organ would respond to a surgical tool.
- The virtual object model must be able to represent the "inner matter" of an object and not just the surface.

The main goals are that the prototype to a high degree holds the following qualities:

- Conceptually clear and easy to comprehend
- Easy to alter and expand

1.3 Structure of the report

This report is structured as follows:

- *Chapter 1, Introduction* presents the domain, problems and objective of the master thesis.
- *Chapter 2, Background* gives an overview of the background theory of the field of surgical simulators.
- *Chapter 3, Related research* presents different research related to the objective of the master thesis.

- *Chapter 4, Conceptual design* describes and discusses the overall architecture and design in accordance with the requirements.
- *Chapter 5, Implementation* documents and substantiates the implementation of the prototype.
- *Chapter 6, Discussion and conclusion* evaluates the work in relation with the objectives, concludes the thesis and discusses future work.

Chapter 2

Background

In this chapter we present the background theory. First, we name some requirements for a surgical simulator. Then we give an overview over methods and techniques used to design and build such an application.

2.1 Concepts regarding surgical simulators

As mentioned in chapter 1, there has been a vast development of technologies related to surgical simulators over the last years.

In development of surgical simulators, physical simulation has proven the most difficult task. During contact between the user guided virtual tool and the physically modelled virtual object under consideration, visual deformations and haptic feedback must be calculated and presented to the user. Whereas the visual feedback only need an update frequency of about 30 Hz to be realistic [BTB+04], the haptic feedback has much greater demands. Our tactile senses need an update frequency of about 300 Hz if the feedback is going to appear realistic [CDA99].

Organic tissue has a complex anatomy, which has proven difficult to reconstruct in a data model. In addition to the physical modelling, high resolution is required for visual realism. If the material should appear natural, our data model should take into account non-linearity, anisotropy and visco-elasticity [BTB+04].

In order to simulate the physical behaviour of organic objects, there are mainly two different methods used [SHS01]:

Spring-Mass: A common approach is to regard the material as elastic. The most popular way of realising this is through a spring-mass system [WH04]. Objects are then modeled as a discretization of points. The points are connected to its neighbours through springs to simulate interaction throughout the object. This could be fulfilled in several different ways. E.g. if you have a surface model of an object consisting of triangles (which is very common), you could simply model all the vertices as mass points and all the edges as springs. More complex modeling schemes exists to try to improve realism.

Finite element method: For the best possible realism, it is necessary to make models based on the law of physics. Finite element methods (FEM) are often use to get a more realistic simulation of objects [SHS01]. These methods are based on actual stiffness and other parameters of the soft tissue being modelled. These methods are also well approved in other disciplines, like computational fluid dynamics, and are extensively used to model e.g. strain and stress in different materials.

Like with the physical behaviour, there are different ways of modeling the spatial extent of the organic objects. The most common way is to use a tetrahedron mesh due to its good adaptability to any object shape [rKWP04]. It is also often used in a way where the surface is extracted directly as all the tetrahedral faces that is not adjacent to any other tetrahedron. If more regular modeling primitives, like cubes, are used, one normally need to model the surface explicitly, and attach this to the volume model in a consistent way.

As mentioned above, for a simulation to appear realistic, we need to keep the visualisation and haptic rendering loops running at respectively 30 and 300 Hz. An important issue therefore becomes computational power. A way to concentrate the computational effort on what's important, is to use a multi resolution model. Another technique, is to make use of different simulation methods in the area of interest and areas more distant to the interaction.

Cutting in an organ, is an important issue of surgical procedures. To accomplish this, there are some challenges that must be overcome. E.g. the cutting

has an immediate impact on the topology of the mesh representing the organ. This leads to different updating issues, depending on both the spatial modelling and the simulation method used.

To minimize these problems and help keeping the amount of data in a reasonable quantity, some make use of several layers of surfaces, or other ”modelling tricks” to depict a volume [BM02]. This gives you the most important volume properties without having to explicitly model the entire volume.

2.2 Summary

In this chapter we have looked at the genaral background to the field of surgical simulators. We give an overview of the different concepts and subjects in this area. This includes physical simulation methods, ways of modelling the spatial extant of virtual objects, issues regarding computational power, and challanges when cutting in virtual models is to be supported.

Chapter 3

Related research

In this chapter we study previous work on different subjects related to surgival simulators. In section 3.2 and section 3.3 we look in to the to leading ways of representing the physical behavior of virtual objects. In section 3.4 we study multi-resolution techniches, before we in section 3.5 briefly discusses hybrid models. Finally we study cutting techniques in section 3.6.

3.1 Introduction

As mentioned in chapter 1, [BN98] points out three essential components of a surgical simulator, namely *Computer graphics*, *Haptic interface* and *Physical modeling.*

There are countless ways and techniques for visualising our simulation on a computer screen. But it is not the objective of this thesis to explore the different possibilities when it comes to the actual visualisation. We therefore use some common approaches which will be described in chapter 5.

There are also different ways of doing the haptic rendering, but almost all articles that discusses some form of surgical simulation, bases their haptic on a phantom device. As this part is also not the main objective of this theses, we will simply use the phantom desktop and its ”open haptic” interface for this purpose.

When it comes to the physical modeling, there are, as mentioned in chapter 2, mainly to ways of doing this: The spring-mass model which is the easiest to implement and which typically requires less computational power. The FEM models which aim to model more accurate the physical behaviour of an object.

In order to consentrate the computational power where it is needed, a technique often used is multi-resolution. By having a finer mesh representing relevant parts of the virtual object, valuable cpu-time is not wasted on uniteresting partts of the object.

Cutting in a virtual object is a comlicated task and several technique have been developed to achieve this matter. Different techniques are elaborated in section 3.6.

3.2 Spring mass models

[GCMS00] presents a spring-mass system. They use a particle representation defined on a tetrahedral decomposition of their organ model. Each particle is described by its mass, position and velocity. The behaviour of the system is then governed by the well known Newtonian second law:

$$M\ddot{x} + D\dot{x} + Kx = f \tag{3.1}$$

Given a tetrahedral mesh, their particle system is simply obtained by considering each vertex as a particle and each edge as a linear elastic relation between the two particles. Then the mass is distributed among the particles according to the size of the incident tetrahedrons.

[Xav95] describes a spring-mass system that simulates cloth objects. The physics are based on elastically deformable models, but the simulation is improved to take into account the non-elastic properties of woven fabrics. The cloth object is first discretized into a set of masses and springs. In ares of locally high stress, the result becomes non realistic in such simulations. Normally this is solved by increasing the stiffness of the springs that are elongated to much. But this is a method that drastically increases the computational costs. Dynamic inverse procedures are presented as a solution to this problem. The spring becomes a max elongation factor. E.g. 10%. If a spring is elongated more than this, the to endpoints of the spring are move towards each

other such that the spring elongation match the max elongation factor. If one of the mass points are fixed, only the other endpoint of the spring is moved.

In [Mos04] Mosegård presents a method he calls local relaxation (LR) Spring Mass model. This model is based on a static formulation of the Spring mass problem. After evaluating different systems with surgeons, Mosegård made the conclusion that a dynamic simulation does not offer any better realism over a static simulation when regarding surgery simulations. In a static simulation there are no notion of mass, damping or inertia. For each force there is one equilibrium, and to simulate the system this equilibrium is what you calculate. It is done by seeking the configuration such that for internal forces g and external forces f for nodes j and i:

$$\sum_j g_{ij} - f_i = 0, \text{ for all i}$$

That is, when the external and internal forces are in equilibrium for all nodes. The internal forces are calculated as linear springs (default length l and spring stiffness k):

$$g_{ij} = k_{ij}(l_{ij} - \|x_i - x_j\|)\frac{x_i - x_j}{\|x_i - x_j\|}$$

In [VS02] Vassilev and Spanleg presents a volume preserving mass spring method for real time simulation of deformable solids. They introduce a new type of spring, named support-spring which models the "matter" inside the object. This preserves the object's volume without the need of explicit volume calculations during simulation. To achive this affect they model such a spring from all points in the object to the objects centre. If the sum of the length of all these springs at a given time is less then at start of simulating, the springs induces a force outwards on all the points. And opposite if the sum is greater than at start of simulation.

3.3 Finite element modelling

When simulating dynamic behaviour, spring-mass and other adaptive approaches may result in unwanted artifacts in the models behaviour.

Debunne et al. [DDCB01] presents a finite element multi resolution model for animating dynamic deformations of a visco-elastic object in real-time. The model consist of a continuous differential equation that is solved using a local explicit finite element method. A strain tensor representation of the physical forces allows simulation of large displacements without unwanted physical artifacts. Thus, the main advantage over other methods, is the avoidance of vibration at different frequencies, even if different resolutions are applied.

Serby et al [SHS01] assert that the most accurate procedures for modeling elastic deformations of tissue using the finite element method to solve the governing mechanical equations. In each element positions and displacements are interpolated from discrete nodal values. For every element, the partial differential equation governs the motion of material points of a continuum. A discrete system of differential equations yields [SHS01]:

$$M\ddot{u} + D\dot{u} + Ku = r \tag{3.2}$$

where:

- u is the vector of nodal displacements
- M is the mass matrix
- C is the damping matrix
- K is the stiffness matrix
- r is the vector of external node forces

However, several interventions may require topological changes of the finite element mesh, thus making a non-trivial remeshing step necessary. A method for simulating such interventions is presented in [SHS01]. The central idea is to not introduce new nodes, but to displace the existing ones to account for the topological changes. After the displacement of the nodes/elements, the mesh is homogenized to avoid tiny elements which destabilize the explicit time integration necessary for solving the equations of motion. The soft tissue deformations are determined using a complex, non-linear, explicit finite element model.

3.4 Multi resolution

[GCMS00] presents a multi-resolution (MR) method. The main contribution by Ganovelli et al in this work is their way of utilizing a MR representation of their object. The MR models of the objects are built in a pre-processing step. The actual representation of the object, that is; which part of the object is represented by which resolution, are changed dynamically throughout the simulation.

A framework called multi resolution triangulation (MT) is used which was introduced by [6] and [14] in [GCMS00]. A MR model is built by step by step refining or simplifying an original mesh. As Ganovelli notes himself, the update of the MR data structure introduces some time overhead.

In the finite element model presented by Debunne [DDCB01], a non-nested multi resolution hierarchy of tetrahedral meshes is used to represent the deformable body, since they can accurately fit arbitrary geometry. The local resolution is determined by a quality condition that indicates where and when the resolution is too coarse. It is supposing that the motion of a node can be computed from values (positions or displacements) at neighbouring nodes. As the object moves and deforms, the sampling is refined to concentrate the computational load into the regions that deform the most. In this, the space and time sampling is automatically adapted locally to concentrate the computational effort where and when it is needed the most.

3.5 Hybrid models

Combining a finite element method and the tensor mass method is one way to allow for real time simulate and at the same time allow cutting and tearing. Delingette [DCA99] presents a hybrid model. One model based on linear elasticity theory and one model based on finite elements modeling constitute this hybrid model.

The first model pre-computes the deformations and forces applied on a finite element model. This supports the deformation of large structures in real-time. This model does not allow any topology change of the mesh, thus forbidding the simulation of cutting.

A second physical model is analogous to spring-mass models for linear elasticity. This model is based on a dynamic law of motion and allows volumetric deformations and cuttings. However, it has to be applied to a limited number of nodes to run in real-time.

Those two approaches combined into a hybrid model allows real time deformations of large enough anatomical structures. This model is, however, only valid for small displacements.

For this reason the above mentioned hybrid model is further improved by incorporating non-linear elasticity [PDA01]. This property improves the realism of the deformations and solves the problems related to the shortcomings of linear elasticity. The author also addresses the problem of an isotropic behaviour and volume variations by adding incompressibility constraints to the model. This solves the problem of rotational invariance of deformations and takes into account the incompressibility properties of biological tissues. Delingette concludes that this adds realism to the model by improving its bio mechanical realism and thus increases its impact in the learning and training processes.

3.6 Cutting

Cutting is an important aspect of surgical simulation. There are a lot of different ways to attack the relatively difficult problem of cutting. In [BSM+02] Bruyns et al. presents different cutting techniques and group them after how they handle different tasks. [Søb05] has made a good survey of this.

Definition of the cut path: The easiest way of doing this is finding the starting- and endpoint of the cut. Then you open the edges between these to points. Another approach would be to directly trace the virtual tool as it is dragged through the object. The chosen method depends of the wanted/needed realism.

Primitive removal and re-meshing: A simple method for this is to simply remove all adjacent faces to the cut-path. This is however not very realistic, and also here the method chosen depends on the needed realism. A more realistic approach here would be to split the faces adjacent to the cutting path.

Number of new primitives created: [BSM+02] states that existing cutting techniques handle remeshing by either:

(a) disallowing new primitives,

(b) allowing unnecessary new primitives, or

(c) creating a minimal number of new primitives.

where one simply select a subset of the mesh traversed by the tool to represent the cut mesh.

When re-meshing is performed: Generation of new primitives can be done at different times during a cutting procedure. You can generate new primitives while the tool and the primitive are in intersection, after the tool is no longer intersecting the primitive, or after the tool has changed direction. It is easiest to make the new primitives after the cutting is ended, but it is visually more realistic to generate new primitives on the fly, as the cutting propagates.

Representation of the cutting tool: The most common representation is a single point. Other representations can be line segments, triangles or shapes consisting on several primitives.

A cutting procedure that do not produce any new faces is presented in [LD04]. This is achieved by snapping nodes lying on edges intersecting the cutting trajectory, to the cut path.

After initial collision of tool with the organ, the nearest vertex is snapped to the collision point. As the cut progresses the vertices nearest to the intersection point of the tool with the underlying polygon edge are snapped to the tool path.To approve visual realism they have implemented a local refinement of the mesh surrounding the intersection of the tool.

Finally they create a "cutting gutter" to make the illusion of cutting in a volume model. They simply create new triangles from the cutting edge into the object. For the physical simulation they use a mesh free method known as the point collocation-based method of finite spheres developed for minimally invasive surgery.

[ZSJ] shows a cutting algorithm that allows generation of new faces. They have different states of a cutting procedure. Where the cutting starts and

ends, only one edge of the primitive (triangle) is intersected and one algorithm for subdividing such a triangle is called. All the triangles in between the start and end triangle have two of their edges cut, and an algorithm for subdividing such triangles is than called.

Also in this article new triangles are generated inside the cut to give an impression of cutting in a volumetric mesh. The article also discusses how to join to separate cuts (e.g. a cut looping back to its starting position).

In [ZSJ04] Zhang et al. present a an improvement to their above presented approach. Here a local refinement technique is used as they cut through a triangle. The triangle intersected by the virtual tool is deleted from the mesh and replaced by a "patch" consisting of tree smaller triangles. Then these triangles are divided according to how the virtual tool propagates through the patch. This interim subdividing gives a much more realistic impression of the cutting, as the cut much more gives the impression of being opened as you cut. When the tool trajectory has passed through the hole originally triangle, the patch is replaced by a single cut triangle. This also discusses how an object can be split into two or more separate objects if one cuts completely part the originally object.

[BM02] presents schemes for real-time generalized mesh cutting in objects of arbitrary topology. They present a method for cutting in single surface models, multiple surfaces models and a method for cutting in hybrid models in addition to methods for volumetric models. This article also discusses the use of different virtual tools consisting of one or more sharp edges. The cutting occur against these sharp edges. The sharp edges of the tools are enclosed by bounding boxes, and collision detection is performed against the part of the object contained in this bounding box. The decision whether to cut a primitive is dependent on its state. These states are stored as information in the primitive class and used during re-meshing. For example, if the primitive has not been intersected before, then the intersection is recorded and the primitive is said to be in the start state. If at the next iteration the primitive is still in collision, then it is thought to be in the update state. In subsequent iterations, if the primitive is no longer in collision, then it is said to be in the move state and the primitive is cut based on the configuration of face and edge intersections that have been stored previously. In this way the primitives are cut as the virtual tool is dragged through the object.

3.7 Summary

In this chapter we have presented previus work on the field of virtual reality for surgical simulators. First we looked into the two major methods for simulating the physical behavior of virtual objects, namely Spring mass and FEM. Secondly we looked into techniques for applying multi-resolution to a geometrical representation before we briefly discussed hybrid models. Finally we elaborated the field of cutting techniques.

Chapter 4

Conceptual design

In this chapter we describe the architecture and design of our prototype. Section 4.1 give an overview of aspects influencing the design. Section 4.2 discusses the choice of geometrical representations and its implications. In section 4.3 the concept of multi resolution is presented and the reasons for incorporating that technique is given. In section 4.4 the different possibilities for modelling physical behavior of an object is given, and our choises are substantiated. Finally ctting is discussed in section 4.5.

4.1 Introduction

Many different models for representation of virtual objects and simulation of their physical behavior have been proposed. Finite element methods (FEM) and spring-mass models are most commonly used for the physical simulation [SHS01]. Tetrahedron are most often used for the graphic representation [rKWP04]. For surface models, triangle meshes are the most used modelling method. To be able to model the inside of objects it is important to use a volume-based model, or several layers of surfaces to make an illusion of volume. This is of course also a necessity if cutting is to be implemented.

To have a model that is conceptually easy to understand and work with, [rKWP04] suggests a cube based approach instead of tetrahedron, due to its simplicity. This geometric structure also lends itself very well to implementing multi resolution. Similar to most others, [rKWP04] does not discuss their

implementation any further than to the conceptual level.

However, to assure a clear code-base to work with, the actual implementation is also of great importance. It does not help to have an elegant and understandable conceptual design, if the implementation makes it difficult to maintain clarity and a good overview.

The choice of physical simulation method and data structures also are important factors that influences how good the over all concept becomes with respect to simplicity and maintainability. Design philosophy of course also plays an important role.

To help keep a good overview and to achieve the maintainability and extensibility, an object oriented design will be used. This is a well documented and tested design philosophy to handle great complexity.

As [rKWP04], we adopt a cube based octree representation as our architecture. This a architecture lends itself very well to object orientation. Each cube consists of eight vertices which are displaced when exposed for forces, thus animating the object. An overview of this architecture is provided in figure 4.1.

4.2 Geometrical representation

As described above, the geometry of the object is represented as an octree in this thesis. Initially, the root node represent the entire object. Then the root node is split into eight children. Each of these new cubes are again split, and so on, until one reaches the wanted starting resolution of the object. At the end of this process the object is represented by all the leaf-nodes of our structure. That is, all cubes that have no children.

Each cube will have references to the vertices of which it consists of. Contrary each vertex have references to each cube of which this vertex is a corner. The need of the references in a cube to its corners will be explained below in section 4.2.1.

The references in a vertex to all the cubes it is a part of, are needed for a vertex to find all its neighbouring vertices. It is also needed when a vertex is split, which will be further described in section 4.5.

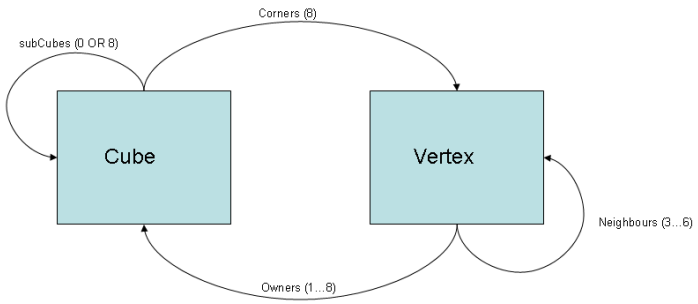


Figure 4.1: Overview of the architecture

The octree structure assures the possibility for implementing multi-resolution without destroying the data structure and also makes it easy to implement both depth first and breadth first traversal and search through the data structure.

An alternative way to design the cube structure would be to also introduce the concept of faces in our architecture. Each cube would have six faces and each face have four corners. This would not contribute much to clarity however, and would actually create more complex functions for building the data structure. This is due to the fact that faces would normally be shared by two cubes, and vertices would be shared among several faces. With only vertices directly under cubes in the hierarchy, we only have one level of "sharing". When the faces are needed, e.g. when drawn by OpenGL, the faces are given implicitly through the different corners of the cube.

4.2.1 Cubes

A cube consists, as explained in the start of this section, of its eight corners. A cube can have anywhere from three to six neighbours. A corner cube of the

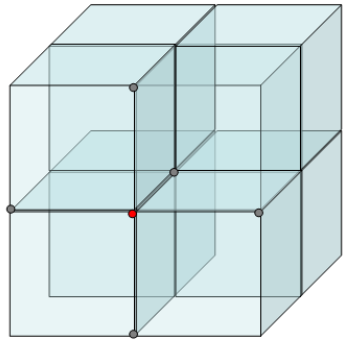


Figure 4.2: Vertex (red dot) with its neighbours (grey dots)

structure would have three, whearas a cube totally surrounded by other cubes would have six neighbours. A cube have a list of references to its corners. These references are as told passed on to its children when the cube is split in 8 child-cubes. If it had not the references to its corners, these vertices would have to be instantiated again, when the children were generated. It is essential that a vertex is actually the same instance by all the different cubes that has this vertex as one of their corners. How exactly to achieve this we'll get back to in chapter 5

4.2.2 Vertices

The corner of the cubes are named vertices. The vertices are the main part of the physical simulation. In our model each cube represent a discretized element of the object. Its movement as the object is simulated, is realized through moving each of its corners. The vertices will have a variable holding its position and and a variable holding possible extern force influencing them. Each vertex has from three to six neighbours. See figure 4.2

When simulating the object, the vertex is influenced from all its neighbours as well as from extern forces. The forces influencing the vertex will be covered in section 4.4.

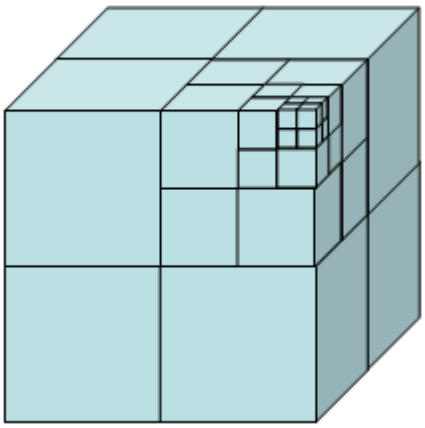


Figure 4.3: Multi-resolution. The nearest, upper subvolume is expanded 3 levels

4.3 Multi-resolution

To be able to model the geometry with appropriate solution without getting to many nodes, we adapt the multi-resolution technique from [rKWP04].

A complete model for virtual surgery training should be realistic, interactive, and should enable the user to modify the topology of the simulated objects. In order to ensure the required speedup and to support dynamic changes of the topology due to cuts of the represented tissue, a multi-resolution technique is adapted. In doing this, time used for simulation and rendering is saved by representing at high resolution only the object parts considered more important or critical. The general idea underlying the multi-resolution technique is that any multi-resolution mesh can be built through local operations that progressively modify an initial mesh through refinement. The original cell, in our case a cube, is split into child cells. Figure 4.3 illustrates a cubestructure with cells split into deifferent levels.

4.4 Physical simulation

To simulate the interaction with the organ that is represented, a method to model the physical behaviour of the object is needed. As stated in chapter 2, there are mainly two different methods for realising this. A Spring mass model or FEM could be implemented. As the Spring mass model is conceptually easier to understand and also easier to implement, it is a good choice when aiming for maintainability and extensibility. [Mos04] states that the surgeons do not need the information from a time dynamic simulation, so a static simulation is sufficient. This leads to his formulation of the LR Spring mass model as described in chapter 3. In such a system the solution can be found by the following algorithm presented as pseudo code:

```
Repeat until time δ has elapsed
    for every i∈ {1, 2, ..., n}
        (a) F_i = Σ_j g_ij + f_i
        (b) x_i = x_i + αF_i
```

where $\sum_j g_{ij}$ is the sum of the forces on i from all of i's neighbours, and f_i is the external force on i, i.e. the force from the virtual tool.

For every particle we find the vector representing forces onto the particle. The particle is then displaced along this vector. As [Mos03] explains:

> *The algorithm looks very much like an Euler integration without the velocity variable. The constant is in many ways like the step size of a numerical solution of a differential equation. It must be low enough for the algorithm to converge, but as big as possible for real time performance.*

To apply forces on an object, the forces acting upon the objcet need to be specified. The only force considered is the force from the virtual tool. The haptic API will exhibit functions that return a force vector when touching an object in the scene. The coordinates of the hit are also returned. Our object is composed of cubes. When we recognise a hit, we can localize which cube and which face of the cube is hit. As explained in section 4.2, the faces are not modelled explicitly in our design, but are given implicitly through a cube's

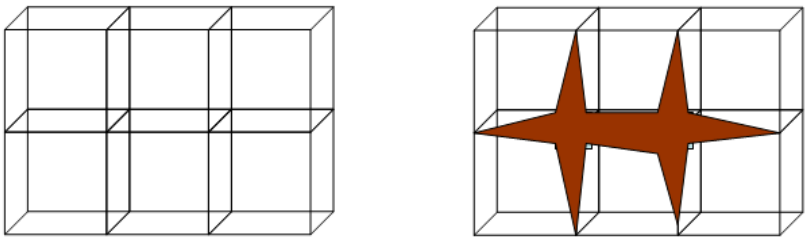


Figure 4.4: Object were two vertices have been cut

corners. Ideally, the force would be weighted on all the four corners of the hit face, according to the distance from the hit point in the face till each of the corners. A simplification however, is just to apply the force to the nearest vertex.

Each vertex stores the external force acting upon it. Under the simulation iteration, the vertex check its distance to all its neighbours and add the force from each neighbour to its force vector. Finally, all the forces are summed and as described in the pseudo code above the vertex is moved a short distance along this vector.

4.5 Cutting

As discussed in chapter 3, there are many ways of cutting in a virtual model. There are methods of different degree of realism and thus different complexity. But all methods are to a certain extent complex, and they all involve quite some computational power.

To realize cutting in a simplest possible manner, and not disturb the regular mesh, an original idea is proposed. The cubes are attached through the vertices they share. If the resolution is high enough, implementing cutting could be achieved through detaching the four cubes adjacent to the nearest vertex of the cutting tool. This is achieved by having each cube instantiate a new vertex instead of the one being cut. This will be thoroughly explained in chapter 5.8. The cubes would obviously be detached only in the one vertex nearest the tool. In figure 4.4, vertices that have bin “cut” in this manner are illustrated.

To avoid the procedure implementing cutting being run several times, we simply mark each cut vertex as cut, and test this property before executing the cutting algorithm.

All organs have an intern tension which causes an incision to gap when cut. The idea is that this tension can be modelled by having each vertex experience a "positional force" that works in such a way that if the vertex is moved away from its original position, it acts a force on this particle in the direction of its original position. Then you can initialise the model so that the spacing between its nodes is greater then the virtual springs constituting the physical modeling of the object, and thus the tension is achieved.

When a vertex than is split, the new vertices being created, will be attracted to their neighbours. Thus, the incision would open up as supposed.

This "positional force" also makes sure the model do not drift away when touched.

4.6 Summary

In this chapter we have discussed the conceptual design of the prototype to be implemented. The focus have been on the overall architecture of the datastructure, which will be based on an octree structure. The main objects in the design are cubes and vertices. The principles behind the physical simulation, how to achieve multi-resolution, and in which way cutting is to be handled, have also been elaborated.

Chapter 5

Implementation

In this chapter, we describe the actual implementation of our design and substantiate the major choices. The implementation of all main parts and procedures will be thoroughly documented, and actual code example will be shown where adequate.

5.1 Introduction

The code is written in C++ using Microsoft Visual Studio.NET as IDE. The choice of C++ is natural because this is by far the most used programming language when it comes to visualization and haptic programming. The OpenGL and OpenHaptics api's are also implemented in C++ and thus makes it an advantage in itself using C++.

As stated in section 4.1, the thesis adopt an object oriented design and thus it is natural to use an object oriented programming technique.

5.2 Haptic interface

A phantom desktop, delivered by Sensable Technologies Incorporated, is used for haptic output. The phantom desktop has six degrees of freedom. In addition to 3 degrees of freedom positional sensing, it reflects the roll, pitch and

Original OpenGL code	OpenGL code for haptic rendering
	hlBeginFrame() hlBeginShape(SHAPE_TYPE)
glBegin(GL_QUADS) glVertex3f(0,0,0) glVertex3f(1,0,0) glVertex3f(1,2,0) glVertex3f(0,2,0) glEnd()	glBegin(GL_QUADS) glVertex3f(0,0,0) glVertex3f(1,0,0) glVertex3f(1,2,0) glVertex3f(0,2,0) glEnd()
	hlEndShape() hlEndFrame()

Table 5.1: OpenGL vs OpenHaptics

yaw of the virtual tool it guids. The device is capable of 3 degrees of freedom positional force feedback whereas the rotation (roll, pitch and yaw) have no feedback.

OpenHaptics™ is chosen as api towards the haptic device. This is an open source api, also delivered by Sensable. The choice of OpenHaptics is mainly due to its perfect match to the OpenGL api. The OpenHaptics api uses OpenGL commands for capturing the geometry and it is therefore easy to reuse the OpenGL code for the haptic rendering. Table 5.1 illustrates this with a banal example.

The OpenHaptics api is divided in two parts. The first one is named Haptic Device API (HDAPI). This is a low level api where you have total control of all functionality of the haptic device. You may for instance render the forces directly to the device. The other part, the Haptic Library API (HLAPI), is on the other hand a high level api which hides much of the low level functionality. E.g. force equations and thread handling. Obviously it is the HLAPI which is designed familiar with OpenGL, and thus will be used in this work.

The collision detection is also done by open OpenHaptics. It is not described in detail, but there is two different methods available. The choice of method is decided by the call to `hlBeginShape(...)`. The alternatives are HL SHAPE FEEDBACK BUFFER and HL SHAPE DEPTH BUFFER. When the feed-

back buffer method is used OpenHaptics uses OpenGL's feedback buffer to capture the geometry. Here all primitives of the geometry are stored, so you can feel the hole object independent of what camera view you have. E.g. you can also feel the backside of the object. This would obviously be the easiest choice, as we want to be able to feel the hole object without having to change the visual view. This method has however a bug, and this causes the virtual tool to "fall through" the object between its adjacent faces.

To accommodate this problem it is possible to use a feature called "haptic camera view." When enabling this feature OpenHaptics sets up a camera additional to the one used for the visualizing. This haptic camera follows the viewpoint of the virtual tool, and thus enables us to touch the backside of an object even with the HL SHAPE DEPTH BUFFER method. Accordingly we use the HL SHAPE DEPTH BUFFER method with the haptic camera view enabled.

To make the collision detection more effective, we should not all the time check the whole geometry for collision. Rather it should be made a selection of the primitives, based on a bounding box algorithm, or other approaches. It was not time to implement this efficiency improvement, but it will be briefly discussed in chapter 6.3.

5.3 Cubes

The class `Cube` is implemented to represent the octree structure. Cubes are represented as eight vertices. To realize the hierarchical structure, the `Cube` class has a member variable named `myChildren`. This is a pointer to a struct holding pointers to 8 instances of `Cube`. These 8 instances of `Cube` are referred to as the relevant cube's children. To generate its children, the `Cube` class has the member function `Split()`. To achieve not having different instances of any vertices, the parent cube generates all vertices needed by its children, and passes on the references.

To generate the 19 extra needed vertices when a cube's children are generated, it is iterated through the x- y- and z-dimensions of the cube. During this iteration the vertices are given positions half way between the parent's corners. This is illustrated by the code in example 1.

Pointers to all the 27 different vertices needed for the cube and its 8 children,

are already stored in the cubes member variable `cv`. `cv` holds a tree dimensional array of pointers to vertices. The eight corners are already initialised, so what's needed is to initialise the other 19.

These 19 uninitialized vertices in a cube could also be used in correcting an artifact coursed by adjacent cubes of different resolution. This possibility will be discussed in chapter 6.3.

Example 1 Initialising vertex positions (fragment of the `Split()` method)

```
dimOrg = cube.size;

for (int k=0; k<3; k++){
  for (int j=0; j<3; j++){
    for (int i=0; i<3; i++){
      //Internal Vertices (not one of the corners)
      if ( i==1 || j==1 || k==1){
         cv->v[i][j][k]->origPos.x = cv->v[0][0][0]->origPos.x + float(i)*(dimOrg/2);
         cv->v[i][j][k]->origPos.y = cv->v[0][0][0]->origPos.y + float(j)*(dimOrg/2);
         cv->v[i][j][k]->origPos.z = cv->v[0][0][0]->origPos.z + float(k)*(dimOrg/2);
      }
    }
  }
}
```

When the system is initialised, the cubes are split to a certain level, giving the standard resolution for the whole object. This is achieved through a test in the constructor of the `cube` class. This is implemented by the code fragment in example 2.

Example 2 Generating subcubes

```
//Calling split method if the cube is larger than default resolution
if ( abs(corners->v[0][0][0]->origPos.x - corners->v[2][0][0]->origPos.x) > 1.0 ){
  split();
}
```

5.4 Vertices

The Vertices are implemented trough the class `Verteks`.[1] As mentioned in chapter 4, the vertices have from three to six neighbours. To keep track of these neighbours, the `Verteks` class has a member variable `vNeig`, which is a list of pointers to vertices. To find its neighbours, the `Verteks` class has a method `AddNeighb()`. This method gather all the vertex' neighbours from the vertex' cubes, and add them to the vertex' neighbour list `vNeig`. Duplicates must be avoided during this procedure. Example 3 shows the code implementing `AddNeighb()`.

Example 3 Building the neighbour list

```
void Verteks::AddNeighb()
{
  /*each call to a cubes RetVertN()
  returns all three neighbours of the calling Verteks in this cube*/
  Verteks** tmp;

  for(int i=0; i<myCubes.size(); i++){
    tmp = myCubes[i]->RetVertN(this);
    for(int j=0; j<3; j++){
      vNeig.push_back(tmp[j]);
      for (int k=0; k<(vNeig.size()-1); k++){
        /*If the vertex is found in the list before the last element,
        it is popped off again*/
        if ( vNeig[k] == tmp[j] ){
          vNeig.pop_back();
          break;
        }
      }
    }
  }
}//void Verteks::addNeighb()
```

When traversing the tree structure we end up with a lot of duplicates of the vertices, because many cubes share the same vertex. To avoid having to check

[1] It seemed like the string Vertex overlapped with some reserved words, so Verteks was chosen :-) !

for duplicates during the simulation iterations, which would be far to time consuming, a global list of the vertices is built. This list is built when generating the tree structure, in the `Split()` method. Each time a new vertex is given its original position, it is added to the global vertex list if it is not there before. This is shown in example 4. This code fragment is run directly after the three lines of code in example 1; inside the same triple for-loop.

Example 4 Building global vertex list

```
int h;
/*If the Verteks already exist we want it to be the same!*/
for (h=0; h<incVertList.size(); h++){
  if ( cv->v[i][j][k]->IsEqual(incVertList[h]) ){
    cv->v[i][j][k] = incVertList[h];
    break;
  }
}
/*Than it wasn't in incVertList from before and we add it*/
if( h >= incVertList.size() ){
  incVertList.push_back(cv->v[i][j][k]);
}
```

5.5 Simulation

The simulation is implemented in the `Mesh` class through the method `Simulate`. This method does nothing else than what is illustrated in the pseudo code shown in chapter 4.4. It iterates through all vertices and call the method `Move()` in the class `Verteks` that calculates the forces influencing the relevant vertex. After the forces are calculated the vertex moves itself. The `Move()` method is given in example 5

The `SumNeigForce()` method calculates the force from all the neighbours to the vertex under consideration. If the distance from a neighbour is less than original, a vector from the neighbour towards the relevant vertex is calculated. An opposite vector is claculated if the distance is greater. This vector represent the force from the actual neighbour acting upon the relevant vertex.

The force vectors from all the neighbours are then summed to get the resulting force from all the neighbours combined. This force vector is finally scaled to

Example 5 Method for applying forces onto a vertex

```
void Verteks::Move()
{
        float extFFac = 1.0;

        hduVector3Df neighb = SumNeigForce(); //Force from neighbours
        hduVector3Df nail = GetNailForce();   //Force from origo

        ext_force *= extFFac;
        hduVector3Df tmpF = ext_force + neighb + nail;

        pos.x += (float)tmpF[0] * alfa;
        pos.y += (float)tmpF[1] * alfa;
        pos.z += (float)tmpF[2] * alfa;
}
```

get the appropriate magnitude of the force. Equivalent the "positional" force is calculated in `GetNailForce()`.

As seen in example 5 , the three factors *extern force*, *neighbour force* and *nail force*, are summed to get the total influence on the vertex, and the vertex is then moved according to this vector.

5.6 Traversing and searching

A convenient quality about a tree structure, is that it lends itself very well both to a depth first traversal (DFT) and a breath first traversal (BFT) of the data structure. Implementing a depth first traversal is extremely simple and is illustrated by the pseudo code in example 6

In this prototype a DFT is used both to rebuild the vertex list after a cube is split in a multi resolution action, and when the cubes are drawn by OpenGL.

A depth first *search*(DFS) is used to get the hit cube, or the nearest vertex of the hit cube, when the virtual object is touched. This search is only a small extension of a DFT. Code for this is shown in example 7.

As seen, the only functionality implemented on top of the traversal algorithm,

Example 6 Depth first Traversal

```
//recursive depth first traversal
recursive_DFT(rootNode)
{
  //If node is not a leaf node
  if (!root->isLeaf){
    for (int=0; int<numChildren; i++){
      recursive_DFT(children[i]);
    }
  }
  else {
    Do your matter;
  }
}
```

is a help function to check if the coordinates passed by `poxyPos` is indeed in the current cube. We don't use a BFS in this prototype, but such an algorithm could be used to speed up the simulation, which will be discussed in chapter 6.3.

5.7 Multi resolution

When a cube is touched, it is required that this is split in eight children. Possibly these eight children can be split again, and so on, depending on how many levels of subcubes that is wanted. To implement multi resolution, some actions have to be taken in addition to calling `Split()` on the relevant cube. First the cube must be removed from all its vertices' `myCubes` list. Then `Split()` is called. The method implementing this is `Cube::MultiRes(int l)`. As seen this method takes one parameter, `l`. `l` guides how many levels of subdividing is to occur. So if `l` is greater than one, it is decremented by one and `Cube::MultiRes(int l)` is called on all the eight generated subcubes. `Cube::MultiRes(int l)` is shown in example 8.

After `Cube::MultiRes(int l)` is run, the global vertex list has to be cleared and built again, to get a consistent vertex list. Finally all vertices have to add their neighbours again. This is achieved through calling `vertekesList.clear()`, `MakeVertList(meshRoot)` and `VAddNeighbors()`. Figure 5.1

Example 7 Depth first Search

```
//Recursive depth first search
Cube* Cube::GetHitCube(hduVector3Dd proxyPos)
{
  if ( !inCube(proxyPos) ){
    return NULL;
  }
  //Is in Cube!
  Cube* myRet;
  if (!isLeaf){
    for (int k=0; k<2; k++){
      for (int j=0; j<2; j++){
        for (int i=0; i<2; i++){
          myRet = myChildren->cubes[i][j][k]->GetHitCube(proxyPos);
          if (myRet != NULL){
            return myRet;
          }
        }
      }
    }
    return NULL;
  }
  else {
    return this;
  }
}
```

Example 8 Multi resolution

```
void Cube::MultiRes(int l)
{
  //First we remove this cube from all it's vertices' myCubes
  for (int k=0; k<3; k++){
    for (int j=0; j<3; j++){
      for (int i=0; i<3; i++){
        cv->v[i][j][k]->RemoveCube(this);
      }
    }
  }

  //then we call split() to split the cube
  split();

  //If l is greater than one, we call MultiRes(...) on all the generated subcubes
  if (l > 1){
    for (int r=0; r<2; r++){
      for (int q=0; q<2; q++){
        for (int p=0; p<2; p++){
          myChildren->cubes[p][q][r]->MultiRes( l-1 );
        }
      }
    }
  }
}
```

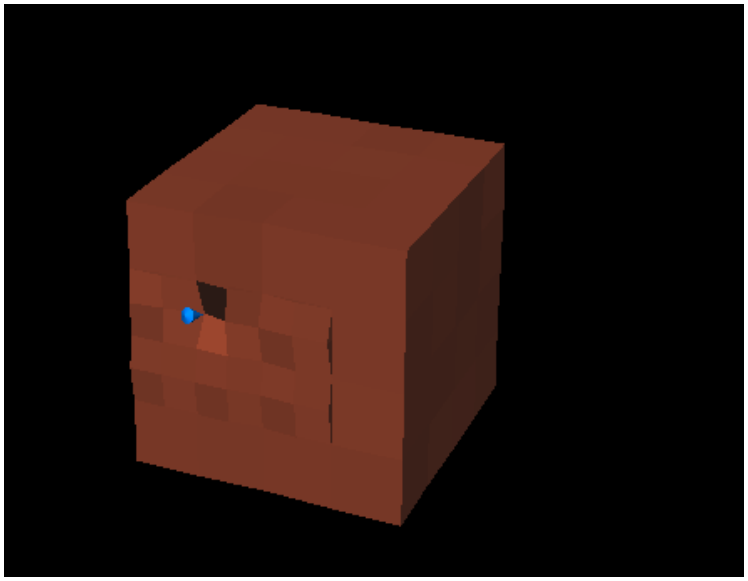


Figure 5.1: Multi resolution

illustrates that some of the cubes in the surroundings of the tool is split one level.

5.8 Cutting

To implement the cutting algorithm, the vertex that is to be cut simply have to get all its cubes to instantiate a new vertex instead of itself.

These new vertices should have the same original position as the one they are replacing. This is realised in the `Verteks` class by a call to the method `Cube::MakeNewVert(Verteks* v)`. This code is shown in example 9. After `Cube::MakeNewVert(Verteks* v)` is run, the same clearing and rebuilding procedure of the vertex list, as was explained necessary after the multi resolution procedure, is needed.

In figure 5.2 we se a cube where two vertices have bin cut.

Example 9 Cutting algorithm

```
void Cube::MakeNewVert(Verteks* v)
{
  /*new Verteks, add this cube to new Verteks'
  myCubes, change reference*/
  Verteks* newV = new Verteks;

  newV->origPos.x = v->origPos.x;
  newV->origPos.y = v->origPos.y;
  newV->origPos.z = v->origPos.z;

  newV->pos.x = v->pos.x;
  newV->pos.y = v->pos.y;
  newV->pos.z = v->pos.z;

  newV->isCut = true;
  newV->myCubes.push_back(this);

  /*Find the correct vertex and change
  the reference to point to the new one*/
  bool found = false;
  for (int k=0; k<3; k++){
    for (int j=0; j<3; j++){
      for (int i=0; i<3; i++){
        if (v == cv->v[i][j][k]){
          found = true; //For debugging...
          cv->v[i][j][k] = newV;
        }
      }
    }
  }
}//void Cube::MakeNewVert(Verteks* v)
```

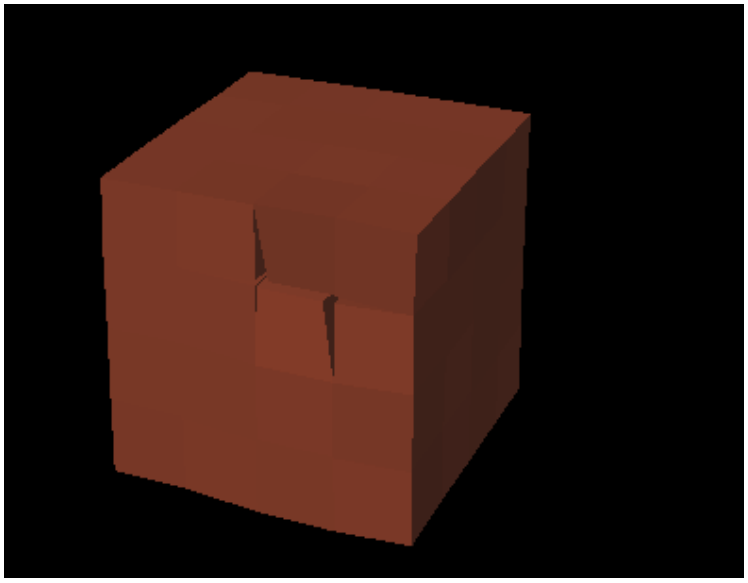


Figure 5.2: Cube having been cut

5.9 Visualisation

The visualisation is as mentioned done by the use of the OpenGL api. Only the most necessary features were implemented to visualise the object. This includes i.a. a single directional light source. A lot of OpenGl features and other techniques could be used to improve the visual realism. This includes more and different light sources, other lightening models, textures and stereoscopic view to name some possibilities. But this was out of the scope of this thesis.

5.10 Summary

In this chapter we have thouroughly documented the implementation. We have described all main algorithms and we have explained the implementation of all vital parts of the protoype. Code examples have been shown were adequate.

Chapter 6

Discussion and conclusion

In this chapter we discuss to what degree our work fulfills the requirements stated in chapter 1.2. In section 6.3 we list recommendations for further work.

6.1 Discussion

As listed in chapter 1, there are four functional requirements for our prototype:

- 3D visualising
- Volume preservation
- Haptic module
- Physical simulation

The visual impression of 3D is present, and the virtual object is shown in a satisfactory way. A lot of improvement could be done to better the visual impression, but as mentioned in chapter 5.9, this was not in the scope of this thesis. Some possible improvements are however listed in section 6.3.

Since the virtuel object is modeled as an octree representation of cubes, the volume preservation is taken care of.

When touching the vitual object by the haptic device, the response is quite realistic. As the tool touches the object, one can feel the resistance which

smoothly increase the harder one pusches. There is no noticable haptic artifacts, like the haptic device juddering.

When applying preassure onto the virtual object, the behaviour of the object is quite realistic. The model deform inwards and when the virtual tool is moved away, the object reshape in a controlled manner. Thus the physical simulation of the object works very well.

As mentioned in chapter 5.2, increasing the efficiency of the collision detection, would be of great importance. This would especially be the case when objects containing more data are to be modelled. For our model however, the simulation runs fast enough to give the user the impression of working in real time.

The cutting functionality works, but not satisfactory. When an incision is made, the cubes are detached in their corners, and the result is a gap between all four cubes surrounding the cut vertex. Accordingly the incision spread out in two perpendicular directions instead of just in the cut direction.

Multi-resolution functionality is also implemented. When the virtual object is touched, the touched cube is split into eight child cubes, thus increasing the local resolution. A parameter guiding how many levels of subdivision are to occure, is implemented in the method triggering the multi-resolution. This technique makes it an easy task to adjust the level of detail.

As formulated in chapter 1.2, the main goals for the prototype are:

- Clearity and comprehendability
- Being easy to alter and expand

The choice of architecture is an octree based cube representation. “Cubes” and “Vertices” are the main objects in this architecture. This design gives a structure with objects reflecting real world objects, promoting overall clarity and comprehensibility of the prototype.

In agreement with our choice of object oriented design, the functionality is as far as possibly included within the relevant objects. This helps avoiding unnecessary dependencies, and thus makes it easier to extend and alter the functionality of the application.

Regarding the simulation, it is accomplished by simply iterating through the vertex list. All vertices calculate the forces acting upon them from their

neighbours, add possible extern force, and move themself acoordingly. This is achieved without implementing any logic finding the neighboursthanks to the vertices' neighbourlist which reflects the "spring" connections between the vertices.

Multi resolution and cutting was implemented to further test the extensibility and changeability of our prototype.

Implementing the multi resolution could easily be done since the logic involved in splitting a cube in eight children, was already implemented in the `Split()` method. Moreover, as the functionality for the different objects are independent of eachother, the call to `Split()` is pretty much what is needed to be done. As explained in chapter 5.7, after a cube is split it is needed to make sure that the vertex list is consistent. This is achieved through three function calls, making the need of extra functionality superfluous.

The cutting procedure was even easier to implement than the multi resolution functionality. As elaborated in chapter 5.8, what is needed is simply to have each cube adjacent to the relevant vertex, instantiating a new vertex instead of the relevant one. Finally, the same procedure that is run after the multiresolution step to assure the vertex list is consistent, is run after the cutting procedure.

Regarding the testing it would have been desirable to include an objective outsider for a more thoroughly testing and evaluation.

6.2 Conclusion

In this master thesis, a prototype has successfully been developed according to the objectives and goals of the study. It demonstrates all necessary concepts needed in a surgical simulator, and the prototype is indeed easy to maintain and extend. The prototype is also conceptually clear and easy to comprehend. This is mainly due to choice of architecture and the choices of objects in the design.

The prototype is far from a surgical simulator and are also not suitable for a starting point to build a full fledged operating environment. It is however, a basic prototype that is suitable for further experimentation and serves as a

foundation for implementing different features in the subject of surgical simulators.

6.3 Future work

Here some recomendations for future work is suggested.

Collision detection: If larger data sets are to be modeled, increasing the efficiency of the collision detection is possibly the single most important improvement of the prototype. This could be done by only sending a subset of the geometric primitives to the rendering engine of OpenHaptics. The selection could be based on what primitive is currently touched by the virtual tool, and then only use the neighbouring primitives for the haptic rendering.

Visualisation: For bettering the visual impression there are a lot of techniques that could be implemented. These includes more and different light sources, other lightening models, textures, bumb mapping and stereoscopic view to list some alternatives.

Visual artifacts: Due to the different levels of resolution of adjacent cubes, a visual artifact arise. This can be seen in figure 5.2. We can clearly see gaps between the cubes that are split in multi-resolution and two of the rightmost cubes of the front face of the object. This is due to the fact that the cubes of the highest resolution have vertices not reflected in adjecant cubes of lower resolution. It would be possible to implement functionality that made cubes adjecant to cubes of higher resolution instantiate vertices halfway between all their corners. These vertices would constitute all possible vertices in adjecant cubes of one degree higher resolution. Then the vertices having the same original position could be combined to one vertex instance.

Breadth first search: In chapter 5.6 it is mentioned that a BFS could be used to speed up the simulation algorithm. During the simulation iteration the vertices are iterated through in the sequence they are inserted in the list. Instead of this procedure, one could implemented a BFS with the vertex being influenced by an external force, as a starting point. By using this new sequence of vertices for the simulation, reaching all the

neighbours of the influenced vertex first would be asured. Consequently the force would propagate through the object in fewer iterations, thus the algorithm would converge faster. Again this means that fewer iterations are needed to get the same quality of the simulation.

Bibliography

[BM02] CD Bruyns and K Montgomery. Generalized interactions using virtual tools within the spring framework: Cutting. In *MMVR Medicine Meets Virtual Reality*, 2002.

[BN98] Morten Bro-Nielsen. Finite element modeling in surgery simulation. *Proceedings of the IEEE*, 86(3):490–503, 1998.

[BSM+02] Cynthia D. Bruyns, Steven Senger, Anil Menon, Kevin Montgomery, Simon Wildermuth, and Richard Boyle. A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *The Journal of Visualization and Computer Animation*, 13(1):21–42, February 2002.

[BTB+04] Jeffrey Berkley, George Turkiyyah, Daniel Berg, Mark Ganter, and Suzanne Weghorst. Real-time finite element modeling for surgery simulation: An application to virtual suturing. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):314–325, 2004.

[CDA99] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, 1999.

[DCA99] Hervé Delingette, Stéphane Cotin, and Nicholas Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *CA '99: Proceedings of the Computer Animation*, page 70, Washington, DC, USA, 1999. IEEE Computer Society.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adap-

tive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, New York, NY, USA, 2001. ACM Press.

[GCMS00] Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3), 2000.

[LD04] Yi-Je Lim and Suvranu De. On the use of meshfree methods and a geometry based surgical cutting algorithm in multimodal medical simulations. In *12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)*, pages 295–301, 2004.

[Mos03] Jesper Mosegaard. Realtime cardiac surgical simulation, 2003.

[Mos04] Jesper Mosegaard. Lr-spring mass model for cardiac surgical simulation. In *Proceedings of Medicine Meets Virtual Reality 12*, 2004.

[PDA01] G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *ICRA2001: IEEE International Conference Robotics and Automation*, Seoul Korea, May 2001. Best conference paper award.

[rKWP04] Lenka Jeřábková, Torsten Kuhlen, Timm P. Wolter, and Norbert Pallua. A voxel based multiresolution technique for soft tissue deformation. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 158–161, New York, NY, USA, 2004. ACM Press.

[SHS01] D. Serby, Matthias Harders, and Gábor Székely. A new approach to cutting into finite element models. In *MICCAI '01: Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 425–433, London, UK, 2001. Springer-Verlag.

[SJ98] R.M. Satava and S.B. Jones. Current and future applications of virtual reality for medicine. *Proceedings of the IEEE*, 86(3):484–489, 1998.

[Søb05] Stig Rune Søberg. Manipulation of biological model, based on a mass-spring-system, using a phantom desktop., 2005.

[VS02] Tzvetomir Vassilev and Bernhard Spanlang. A mass-spring model for real time deformable solids. In *Proceedings of East-West-Vision*, 2002.

[WH04] Wen Wu and Pheng Ann Heng. A hybrid condensed finite element model with gpu acceleration for interactive 3d soft tissue cutting: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):219–227, 2004.

[Xav95] Provot Xavier. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of Graphics Interface*, page 141, 1995.

[ZSJ] Hui Zhang, Payandeh S., and Dill J. Simulation of progressive cutting on surface mesh model. *DRAFT6-08*.

[ZSJ04] Hui Zhang, Payandeh S., and Dill J. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference*, volume 4, pages 3908–3913, 2004.