IMPERIAL COLLEGE LONDON

DEPARTMENT OF BIOENGINEERING

Computer-based multi-sensorial environment for anatomy teaching

Dynamic modelling of the rib cage anatomy during respiration

Mathieu Jacob

September 2008

A Thesis Submitted in partial fulfilment of the requirements for the award of MSc in Engineering and Physical Science in Medicine and the Diploma of the Imperial College

Abstract

Anatomy teaching is undergoing significant changes. Computer-aided learning, including three-dimensional environments, is one of the new techniques that appeared in the medical curriculum. Besides, Haptic devices are now of great interest in the medical field. The objective of this global project is to develop a three-dimensional multi-sensorial environment for anatomy teaching. Learning by seeing, feeling, touching indeed seems really helpful. This project focused in particular on the rib cage anatomy and the dynamic modelling of the diaphragm, the ribs and the lungs during respiration. A ChainMail deformation model was used for the deformation of the organs. The first version of the model was tested by teachers and medical students so as to assess the usefulness of such an application. The first results were good and promising.

Acknowledgments

I would like to express my gratitude to my supervisor, Fernando Bello, and Harry Brenton who were helpful and who allowed me to do this project.

I would like to thank Pierre-Frédéric Villard who offered invaluable assistance, support and guidance.

I would also like to thank the Webmaster of the H3D forum, Markus, for his advice, and Samy Tzaprenko for helping and supporting me throughout the project.

Finally, special thanks to the lung teacher who gave us a highly valuable feedback on the model.

Content

List of Figures

1 Introduction

1.1 Anatomy teaching

Human anatomy is the scientific study of the structure of the human body. It can be divided into gross anatomy, histology, embryology, and neuroanatomy. Gross anatomy deals with the structure and positioning of the bones, muscles and internal organs. Histology deals with the organisation of cells and tissues. Embryology is concerned with the development of an embryo. Neuroanatomy deals with the localisation of the function of the human brain, the spinal chord and the peripheral nervous system.

Anatomy is considered as one of the cornerstones of medical education. There is a false premise that knowledge of anatomy is required only for surgeons. It is actually essential for medicine, dentistry, physiotherapy, radiography, and speech and language therapy (Raftery 2006).

How should anatomy be taught? This question, dealing with the methods used for anatomy teaching, has always been a sensitive issue since the beginning of medical education. Nowadays, anatomists have some preferences when it comes to this topic. Patel et Moxham listed their preferences. In descending order, they prefer to teach with practical lessons using cadaveric dissection, practical lessons using prosection, tuition based upon living and radiological anatomy, electronic tuition using computer-assisted learning, models (Patel et Moxham 2006).

1.1.1 Dissection

Dissection has always been the paradigm of anatomy teaching since the Renaissance. Cadaveric dissection has always been an essential and regular feature on medical training.

Figure 1.1: *Medieval dissection*

The advantages of practical lessons using dissection are now well known. First there is a knowledge acquisition and integration. Students can indeed apprehend the anatomical vocabulary. They become familiar with both the three-dimensional relationships of the anatomy and the biological variation. Secondly, students can acquire new skills. They develop manual dexterity, touch-mediated perception of the cadaver, and competence in the diagnostic and training. Finally, their attitudes are improved. Indeed, they establish the primacy of the patient, it promotes their own professionalism through the direct contact with a cadaver, promotes their teamwork attitudes, and they acquire respect for the physical body (McKachlan et Patten 2006).

On the other hand, dissection also has drawbacks, which is why anatomists are always wondering whether it is a good method of teaching or not. First, the emotional impact can be very important and disturbing. Dissection can involve extreme anxiety, emotional disturbances or, on the contrary desensitisation leading to an undesirable detachment from death. Secondly, there are some health and safety hazards. Dissection can involve exposure to embalming fluid chemicals or infectious diseases. Finally, there are some practicalities and cost issues. A cadaver is difficult to acquire, it is expensive to transport and maintain it.

There is an increased length of time required for anatomy teaching lessons, and there is a shortage of qualified anatomists (McKachlan et Patten 2006).

1.1.2 Prosection

The distinction between the active dissection by students and the observation of dissection has always been clear. Learning by watching is called prosection. Prosection was the mainstay of Renaissance teaching, in the so-called 'anatomy theatres', but dissection came to be seen as a more modern way of teaching than observation.

1.1.3 Living body and Medical Imaging

'Live models were rated superior to using cadavers, especially in demonstrating superficial anatomy and landmarks', according to (Barrows, Patek et Abrahamson 1968). Real living bodies have already been used for the need of anatomy teaching.

Anatomy classes can also include images acquired by medical imaging. It is important for the students to familiarize with two-dimensional images to mentally visualize the corresponding three-dimensional reality. Imaging technologies are always evolving, and it will become less and less expensive to use such tools.

1.1.4 Computer-assisted learning

Computer-assisted teaching is more often being used in the anatomy labs. It can show how useful anatomy is. With new technologies, there has been an explosion of computer-based anatomy material that is made available. It can provide the student with an important additional resource. It usually includes anatomical information, but also allows for the user to test his knowledge.

Users can learn at their own rhythm, in a much more relaxed environment. But anatomists and many others insist on the fact that computer-assisted learning will 'never

fully replace the intellectual, educational and emotional experience afforded to medical students by cadaver dissection and even prosecution' (Paalman 2000).

1.2 Physiology teaching

Physiology emerged as a separate discipline from anatomy in the second half of the nineteenth century. While anatomy concentrates on the structure of the human body, physiology is the study of its mechanical, physical and biochemical functions.

Medicine is always evolving – new drugs, new diagnostic and interventional techniques – so it is more and more important for medical students to really understand the principles of physiology (West 2002). The primary basis of a good medical education will always consist in a good understanding of how the body works.

1.3 Thoracic Anatomy

1.3.1 Diaphragm

Whitelaw was the first to construct a 3D picture of the diaphragm from serial transverse sections obtained with a CT scan. (Cluzel, et al. 2000). The diaphragm can be considered as a sheet of tendon and muscle. It is dividing the torso in two parts. The lungs and the heart compose the upper part, and the liver, the stomach, the kidneys, the intestines, and all other abdomen parts compose the lower part. The diaphragm is dome shaped. A boomerang shaped and very strong tendon composes what is called the aponeurosis – the central tendon.

***Figure 1.2**: 3D Model of a diaphragm*

The heart is attached to the diaphragm via its pericardium, thus following the same up and down movement. The diaphragm is higher on the right than on the left. The right part is higher because of the liver lying just underneath, whereas the right part is lower because of the heart. The diaphragm contains 3 orifices to allow for the passage of the aorta, the vena cava and the oesophagus.

1.3.2 Lungs

Characteristics

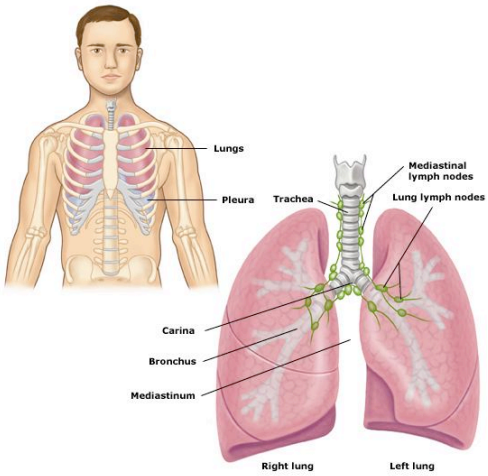


***Figure 1.3:** The lungs and the pleura*

The lungs are the major organs of the respiratory system and are located within the thoracic cavity, which is separated into two halves (one for each lung). These two halves are separated by the mediastinum, which has a membranous structure, contains the heart, the major blood vessels and the trachea (Kaye, Metaxas et Primiano s.d.).

Each lung is divided into lobes. The right lung has three lobes, whereas the left lung only has two lobes. The left lung shares space with the heart. The lung and the chest wall, which contains the ribs, the sternum, the respiratory muscles and the diaphragm, are separated by a very thin layer of liquid called the intrapleural fluid. This fluid occupies the intrapleural space.

The contraction of the diaphragm creates a lower pressure in the chest cavity. Air can thus go into the nose, the mouth, the trachea, and the lungs. When the diaphragm relaxes, its pressure on the lungs increases. Air is forced out of the body.

Lungs are always in contact with the rib cage (ribs, sternum, thoracic vertebrae) and the diaphragm through the pleura. Thus their motion controls the movement of the lungs. (Didier, et al. 2007)

The pleura

The pleura is the thin covering that protects and cushions the lungs. It is made up of two layers of tissue. The parietal membrane covers the chest wall, the mediastinum and the diaphragm. The visceral membrane covers the lungs. A small amount of incompressible fluid fills the space in between. This space is called pleural space.

The fluid allows the lungs to slide easily against the rib cage and the diaphragm during respiration when they expand. The movement of the diaphragm and the ribs during inhalation creates a smaller pressure in the pleural space compared to atmospheric pressure, leading to a tensile stress on the lungs (Didier, et al. 2007).

1.3.3 Ribs

The thoracic skeleton is mainly composed of ribs. Ribs are elastic arches of bone, and a typical human ribcage generally includes 24 ribs, 12 on each side. The first seven ribs – also called true ribs - are connected with the vertebral column and, via costal cartilages, with the sternum. The five remaining ribs – or false ribs – are composed by 3 ribs, whose cartilages

are attached to the one of the rib above, and two floating ribs, which are not attached in front. The space between two ribs is called intercostal space and contains intercostal muscles, nerves and arteries (Gray 1918). The ribs are elongated, flattened, and twisted bones.

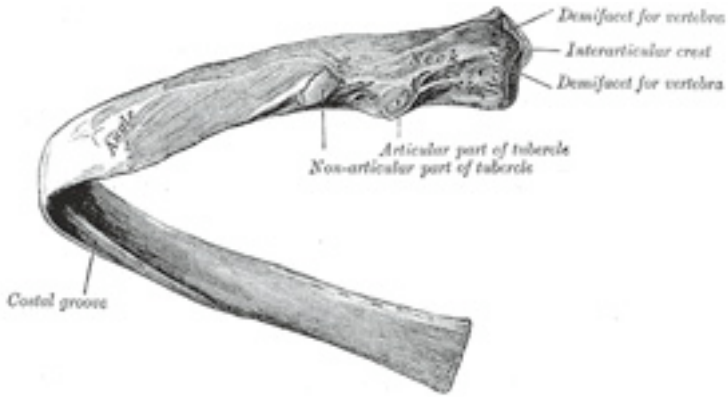


Figure 1.4: *One central rib seen from behind*

1.4 Introduction to Haptics

1.4.1 Haptic technology

Haptic comes from the Greek haptikos (from haptesthai, to grasp, touch) and means the sense of touch. Haptic technology is a technology where the user can feel the sense of touch and control by applying forces, vibration, or motion. The user can thus receive a feedback from computer applications. Such applications include games, virtual reality systems, medicine applications, robotics, and design. With human-machine interfaces and more generally in the computing world, the user is forced to interact with applications by using his eyes. Haptic technology can, in some way, give the user tactile and kinesthetic sensations so that the application looks but also feels like the real world.

1.4.2 Applications

According to the Personal Tech 2008 Top 10 Trends report, 2008 is the year of Haptic feedback. Haptic will be more and more integrated in cell phones and mobile devices. Some cell phones even already have Haptic feedback. For example, buttons are progressively disappearing from cell phones screens, and Haptic technology is able to create a little vibration to tell your finger that you effectively pressed a virtual button.

Including Haptics in computer simulations is still a new technology, but some applications that use Haptic technology are currently being – or have already been - developed, especially in the medical field.

Virtual wheel

CyberForce is a Haptic armature that can apply forces to the hand and arm and that has a positional tracking measuring the translation and rotation of the hand in three dimensions. Many applications are possible. For instance, it can simulate a steering wheel (see Figure 1.5)

Figure 1.5: *The Haptic armature gives the user the feeling of manipulating a wheel*

Haptic suturing simulator

A suturing simulator has been developed by (Haluck, et al. 2001) to teach basic suturing for simple wound closure. It includes a real-time modelling of deformable skin, tissue, and suture material.

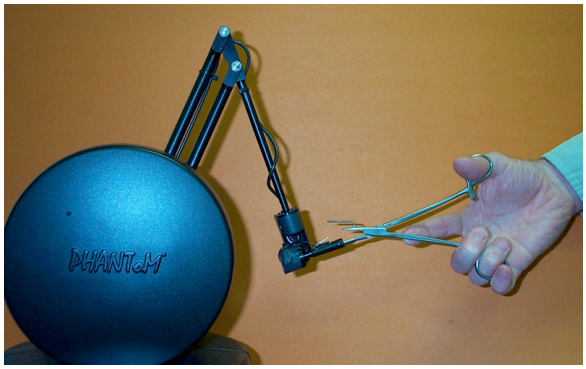


Figure 1.6: *Hemostats are firmly secured to the Phantom end effector by a specially built bracket*

Figure 1.7*: Screen Shot of Finite State Machine Showing a Subset of States and Transitions*

Real hemostats – the needle holders – are attached to the Haptic device. The graphics of the hemostats, needle, suture, and virtual skin are displayed and updated. It will also be possible to measure surgical skills for example. This represents an efficient way of learning the suturing procedure.

Vascular and visceral interventional radiology training

Interventional Radiology is facing many problems. Among them is the risk to the patient during motor skills and cognitive training. An alternative to the current paradigm for training and assessment of core skills is needed. Haptic technology can provide useful tools

to help create virtual environments where the user can train and evaluate his skills. The project CRaIVE is currently developing such solutions. In Figure 1.8, 1.9, two Haptic devices are connected to the computer to simulate needle puncture.

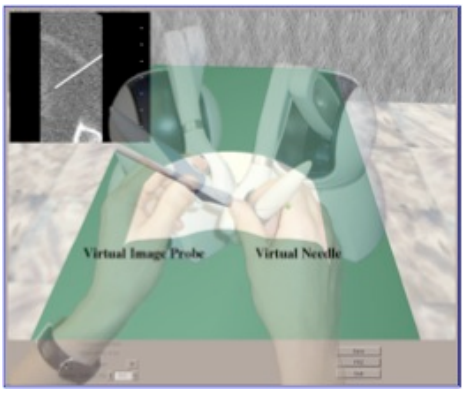


***Figure 1.8**: Screenshot of a needle puncture simulation*
(Copyright: Glyn Davies, Menai Bridge Needle simulation display)

***Figure 1.9**: Two Haptic devices are used to simulate the needle puncture (Phantom)*

1.5 Aim of the project

The objective of the project is to obtain a dynamic model of the rib cage anatomy during respiration, including the ribs, the spine, the sternum, the diaphragm and the lungs. Such models already exist, but the challenge here is to integrate it into an environment, which is compatible with Haptic devices. The aim is to provide a dynamic model that will be either a teaching tool for teachers, or a revision tool for medical student.

2 Related work

2.1 Anatomy

2.1.1 Diaphragm

The central tendon is of constant area (Arora et Rochester 1982). Muscular fibres that originate on the lumbar spine, the bottom edge of the ribcage, and the sternum, connect around this central tendon. Depending on their origins and their insertions, three muscular fibres can thus be identified: the sternal, costal and vertebral muscular fibres.

The sternal fibres are attached to a back part of the sternum: the xyphoid process. The costal fibres are attached to the internal surface of the lower six costal cartilages and their corresponding ribs. The lumbar fibres are fixed to the aponeurotic medial and lateral arcuate ligaments and to some lumbar vertebrae by means of the crura (Cluzel, et al. 2000). These fibres converge to the aponeurosis – the central tendon.

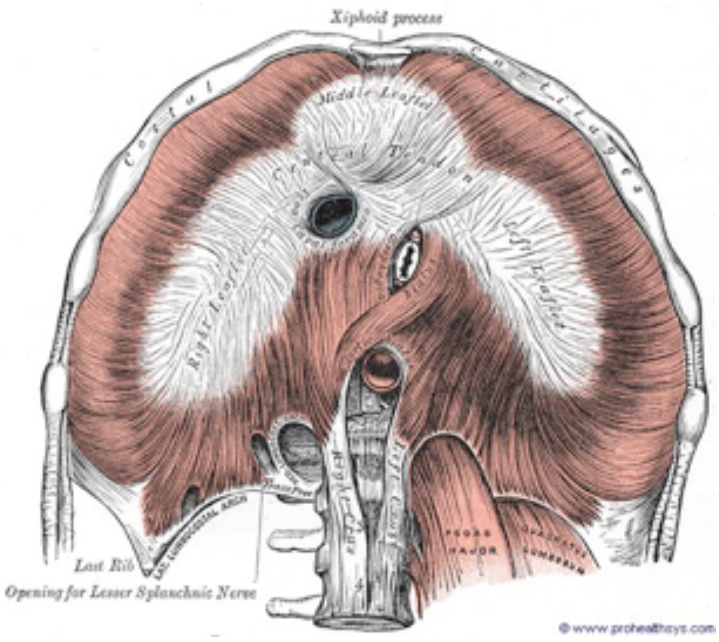


Figure 2.1: *The Diaphragm seen from underneath*

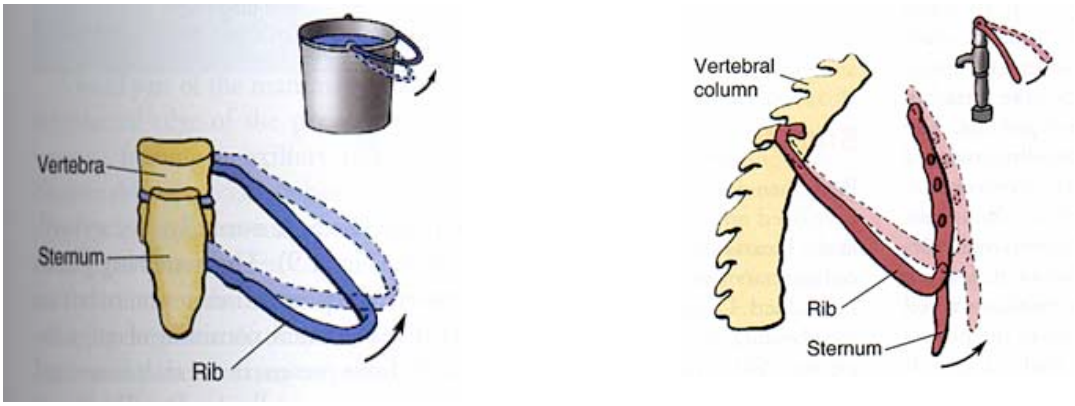
For most anatomists, the diaphragm is the primary muscle of respiration. In other words, it is the prime mover of tidal air. Thus it is traditionally studied as a respiratory muscle. However, it is suggested (Pickering et Jones 2002) that it should be described as two separate muscles: the crural diaphragm and the costal diaphragm. These two muscles are acting in synchrony throughout respiration. But the activities of these muscles can differ during events like swallowing and emesis. However, in this thesis we will consider the diaphragm as one respiration muscle.

During inspiration, the diaphragm muscles contract. It gives the aponeurosis a downward and forward movement. This action drives up the abdominal pressure. This pressure then drives the abdominal content (in particular the liver). The transverse diameter of the chest cavity is also increasing. During this process, the dome is moving in a nearly parallel way to its original position.

During expiration, the diaphragm passively relaxes, returning to its equilibrium state. The displacement of the central tendon during quiet respiration is on average 1.5 and 1.7 respectively in a standing and a supine position.

2.1.2 Ribs

The movement of the ribs can be considered as the combination of a 'bucket handle' movement and a 'pump handle' movement. The 'bucket handle' movement increases the lateral excursion of the ribs, while the 'pump handle' one increases the anteroposterior diameter of the thorax. Thus the rib cage increases in both the anteroposterior and the transverse diameter (see Figure 2.2)



(a) *(b)*

Figure 2.2: *(a) The 'bucket handle' movement (b) The 'pump handle' movement (The Thoracic Cage/Respiration & Breathing s.d.)*

A study (Wilson, et al. 2001) showed that each rib is contained in a plan that has a certain orientation relative with the original coordinate system (x, y, z). They defined a new coordinate system (ξ, η, ζ), where ξ is the intersection of the plane of the rib and the sagittal midplane, η the intersection of the plane of the rib and the perpendicular of the sagittal midplane, and ζ is perpendicular to the plane of the rib (see Figure 2.3). They obtained some data at Total Lung Capacity (TLC) and Functional Residual Capacity (FRC) for the 2nd to the 9th rib. In particular, they obtained values for α and β, which are the angles between the new axis and the original ones. α is called the 'pump handle' angle, and β is called the 'bucket handle' angle.

All the ribs showed a decrease in both α and β with passive inflation to TLC. It seems logic: the plane of the rib has a tendency to coincide with the transverse plane. However, the changes in amplitude decrease. For instance $\alpha_{TLC} - \alpha_{FRC}$ went fro 14.3 ° for the 2nd rib to 6° for the 9th rib, and $\beta_{TLC} - \beta_{FRC}$ went from 13.7° for the 2nd rib to 6.3 for the 9th rib.

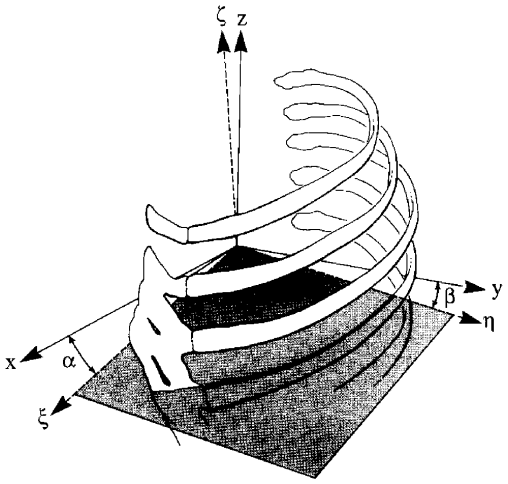


Figure 2.3: *Orientation of the rib plane relative to the transverse plane*

2.2 Dynamic deformation models

Much research has been done in volume deformation algorithms, in order to allow modeling of object deformation. There have been applications in many fields going from surgery simulation, computer games, to animation of muscles, simulation of car accident, or virtual endoscopy.

Two types of deformation models can be defined: geometrically based models, and physically based models. In geometrically based models, the shape of the object is adjusted by changing the characteristics (like the position) of some control point. It can also be adjusted by changing the parameters of an implicit function defining the shape of the object. Generally, these models are fast and have very good performances. Unfortunately, they do not model physical properties of the object, and do not use physical laws to calculate the new state. It is also difficult to provide direct manipulation of an object and intuitive interaction. (Dräger 2005)

Physically based models try to embed the material properties into the model, and physics laws govern the movement. It can thus represent very accurately the real life and the dynamic behaviour of objects. The drawbacks of these models are their high computational cost, and interactivity is not always possible.

2.2.1 Mass-Spring

In a Mass Spring System, the object is represented as a set of points connected together by damped springs. The movement is then based on the forces applied on each point.

This method of deformation is fast and easy. . The major advantage of this method is the simplicity of the mesh. It is thus ideal for direct rendering. The calculation of each step involves differential equation solving, which are easy to compute and to solve. However, the behaviour can become quite unrealistic for large deformations. There can also be some issues when it comes to deal with harder objects such as bone (Dräger 2005). There is also a limited volumetric behaviour because of the local structure of the mesh. Finally, a Mass Spring system has a predisposition to oscillate. This is because it has an iterative structure.

The Mass Spring model is among the most commonly used technique for soft tissue modelling, and although it can suffer from minor drawbacks, its speed and capability is always appreciated.

2.2.2 Finite Element Model

In a Finite Element Model (FEM), the object is subdivided into a mesh of simple elements, such as tetrahedra or triangles. Physical properties, like elastic modulus, stress, and strain, are associated to each element. The corresponding equilibrium equations are solved numerically. FEM has a greater accuracy compared to Mass Spring models, and the results can be very realistic. However, the downside of this method is its very high computational cost. This method is, more than other ones, governed by a trade-off between performance and realism.

Various approaches have been developed to try and overcome this performance issue. The Boundary Element Method solves equations for unknown displacements and forces only on the boundary of the object, rather than the interior as with FEM. The movement is then governed by a large linear system of equations. Such a system could normally not be solved in real time, but with some pre-computations, it becomes possible to obtain at each step a modified system with only few updates compared with the previous system. This allows for fast and accurate deformations.

2.2.3 ChainMail

Another method for fast deformation modelling is the ChainMail. It is a promising approach for soft tissue modeling. A first version of a ChainMail model was developed by Gibson: the 3D ChainMail (Gibson 1997). It was originally created for the deformation of volumetric objects as needed in surgery simulation. The ChainMail model is geometrically based but it is capable of simulating material properties to some extent.

In this model, the object is defined as a set of point elements. The elements are interconnected as links in a chain. So within a certain limit, each point can move freely without influencing its neighbours. When one element of the object is moved and reached this limit, the neighbours are forced to move in a chain reaction that is governed by the stiffness of the links in the mesh. In FEM, there are complex calculations on a small number of elements. The ChainMail model is doing simple calculations on a large amount of elements. There is also a relaxation step in which the energy of the configuration is minimized.

Like the Mass Spring model, the great advantage of this method is its simplicity (Meier, et al. 2005). The volumetric behaviour is guaranteed by the chainlike structure of the model. However, one critical aspect of this model is the resolution of discretization, which makes it hard to use with a high number of elements. The calculation of the deformation for various simultaneous contact points can only be performed at high computational cost.

Gibson has proposed a fast deformable modelling (3D ChainMail). One major advantage of this model is that Gibson proved that each element is processed at most once per time step, but the model only defined for a rectilinear mesh. M.A. Schill et al introduced an enhanced algorithm that is able to model inhomogeneous data (Schill, et al. 1998). With the first method, it was not proven that each element was processed at most once if the mesh was inhomogeneous. The second model solves this issue by using sorted lists during calculation. Li and Brodlie introduced a Generalised ChainMail (Li et Brodlie 2003), which is not restricted to rectilinear grids.

3 Materials and Methods

3.1 H3D

3.1.1 What is H3D ?

H3D is an open-source, scene-graph API (H3D.org - Open Source Haptics s.d.). It is entirely written in C++, and it uses OpenGl and HAPI for graphics and haptics rendering (see Figure 3.1)

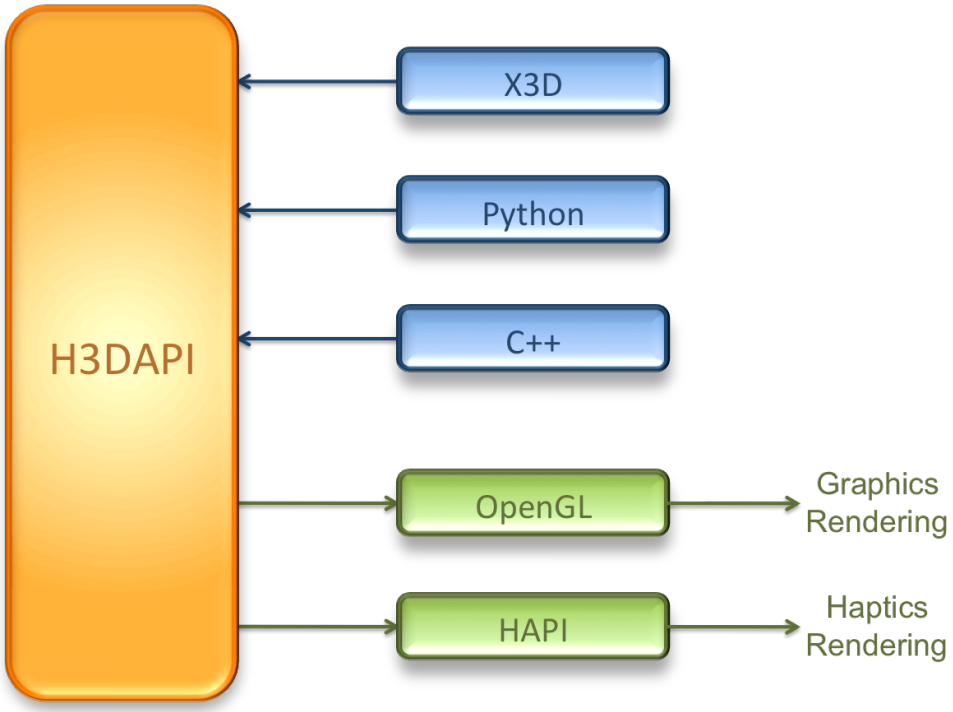


Figure 3.1: *Overview of H3DAPI*

There are three ways of programming applications with H3D: C++, Python, or X3D. A combination of those can also be used.

3.1.2 Features

Standards

H3D is built using many computing standards. The Standard Template Library (STL) – a software library in C++ - can be directly used when programming in H3D. For the graphics, Open Graphics Library (OpenGL) is a common standard and almost all operating system support OpenGL rendering. H3DAPI can be used by programming in C++, but also in X3D. The Extensible 3D file format (X3D) – the successor of the VRML standard – is a standard scene-graph design.

Haptics

As said in 1.4.2, Haptic technology is still new. Thus there are very few scene-graph based APIs that offer Haptic rendering. Using H3DAPI with X3D allows the user to combine the sense of touch with vision thanks to Haptic extensions to X3D. H3D uses the standard library OpenHaptics, and via HAPI can support several other devices. The supported devices include Phantom Devices, Force Dimension Devices, and the Novint Falcon.

Stereoscopic display

H3DAPI supports stereoscopic graphics rendering. It is customized to work with a wide variety of Virtual Reality display systems. It can render several types of stereoscopic display from very standard quad-buffered stereo to hardware display-specific modes.

3.1.3 H3D concepts

Fields

Fields are both data containers and event handling mechanisms. They are the most fundamental concept in both X3D and H3D.

A field, as a data container, can be of any type. By default, the data is simply propagated forward to all outgoing connections. It is however possible to apply some treatment to this data by modifying the *update()* function of the field.

Most of the time in H3D, the type of the fields is *SField* or *MField*, where *SField* and *MField* is a very general type. A typical *SField* is a field that contains one value of some type - *SField* for Single value. A *MField* is a field that contains a vector of values of some type – *Mfield* for Multiple values. For example, a field type could be *SFFloat* if it contains a float number, *SFVec3f* if it contains a vector of three float numbers, or *MFVec3f* if it contains a vector of vectors of three numbers.

Example of fields:

```
ambientColor = "0.6  0.6  0.6"
rotationAngles = "15, 30, 45, 60, 90"
```

In the example, the type of `ambientColor` is SVec3f and the type of `rotationAngles` is MFFloat.

Nodes

Nodes can be seen as field containers and managers, and they are the traditional 'building blocks' of scene-graph APIs. Indeed, in X3D a node represents a XML object and is used to buil the X3 scenegraph. A Node groups some fields together in order to create a larger reusable entity. It determines how fields can be accessed by the user, creates an interface and hides all the internal functionality from the user. Four types of access are defined for a field in a node:

- `inputOnly` : the field is an input to the node, and it is not possible to get its value.
- `outputOnly` : the field is an output from the node, and it is not possible to set its value.
- `initializeOnly` : the value of the field can only be initialized. After initialisation, the field becomes an outputOnly field.
- `inputOutput` : no access restrictions.

Events

In H3D, fields can be connected together with something called routes. It is possible to create a route that goes from field A to field B. When the value of A changes, field B is informed and the field can take appropriate decision. By default, a route between fields of same type will mean that if the value of field A changes, the value of field B will also change to the same value as in A.

Example of route:

```
<ROUTE fromNode="SPHERE" fromField="Color"
 toNode="BOX" toField="Color" />
```

When the value of a field changes, it generates an event that will be send to all the fields it is routed to. The event is then propagated like that until it finds a field without any fields routed from. When a field receives an event like that, its value does not automatically change. It just knows that its value is not up-to-date anymore. If the value of this field is then needed somewhere (a call to the *getValue()* function), it will have to update the value.

3.1.4 Implementation

Create new Fields and Nodes

It is possible to create new Nodes and new Fields for an H3D-based application. The easiest and most flexible way to use them is to create classes in C++ corresponding to Nodes and Fields. These classes are compiled into a DLL library that will be imported in the main X3D-file.

Update Fields

By default, when fields are updated, they take the value of the field they are routed from. To be able to customize the behaviour of a field, it is necessary to specialize the *update()* member function. The best way to understand how to do this is to see the following example:

```
class myFloat : public SFFloat {
   virtual void update()
   {
       H3DFloat f = static_cast< SFFloat *>(routes_in[0])->getValue();

      if ( f  > 25 )
      {
            value = 0;
      }
      else
            value = f;
    }
}
```

In the example, the new field is `myFloat`, its type is `SFFloat` (it can take single float values). `routes_in[0]` is a pointer to the field that caused the event . `value` is the value of the field. In this example the field just copies the value of the input field if it is inferior to 25.

By using the same structure than this example, the programmer can then apply any action to the value of a field when it is updating.

TypedField

An important feature of fields is that it is possible to route together fields that are not necessarily the same types, and to decide what the value will be depending on those other values. The TypedField modifier allows to do that. Let's see how is implemented:

```
class myFloat : public TypedField< SFFloat, Types< SFBool, SFFloat > >{
   virtual void update()
   {
      bool b = static_cast< SFBool *>(routes_in[0])->getValue();
      H3DFloat f = static_cast< SFFloat *>(routes_in[1])->getValue();
      if ( b )
      {
            value = 0;
      }
      else
            value = f;
    }
}
```

In this example, the field is also `SFFloat`, but a `SFBool` field and a `SFFloat` field

are routed to this `myFloat` field. Depending on the value of the `SFBool` boolean, the field gets either the value `0`, or the value of the input float field. In the TypedField modifier, the first term defines the type of the field, and the second argument specifies the routes that are required in input.

Create a new Node

To see how to create a new Node, see the Appendix. The basic structure for a new Node is similar to the structure of the `Chainmail` Node.

In order to create new fields and new nodes in H3D, it is very important to understand the concept of the *update()* member function and the *TypedField* modifier. It gives a first idea on how events are managed. However, it is even more important to understand that even if the *update()* function of a field is well defined, the value of the field will be updated only if it is needed. This can be done either by routing this field to another one whose value is called, or by trying to get the value of this field by calling the *getValue()* function. If this is not the case, the field will receive the event, but nothing will happen.

3.2 Model in Java 3D

A model for the human torso during respiration has already been developed in the past (Bourne 2007). It includes the lungs, the rib cage and the diaphragm. It was not possible to adapt this model to H3D because it was implemented in JAVA with Java3D graphics rendering, whereas H3D is implemented in C++ with OpenGL graphics rendering (see Chapter 3.1). However it gives an idea of what is achievable in terms of deformation and modelling.

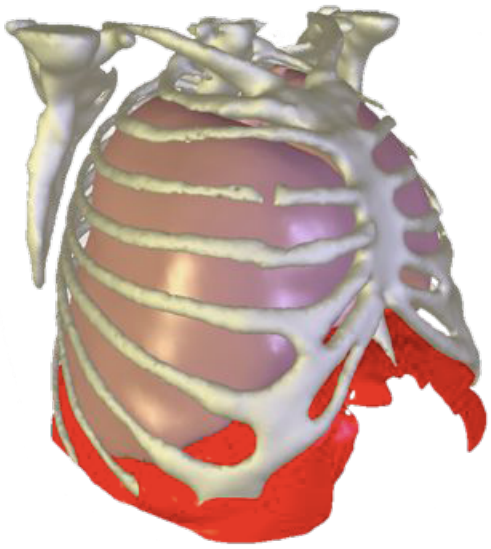


Figure 3.2: *Final result at the end of another project of respiration model*

The diaphragm motion is modelled like a muscle: it contracts and relaxes periodically. The respiration starts here. The movement of the ribs is a rotation. The lungs are in contact with both the diaphragm and the ribs. Their movement forces the lungs to expand like in respiration.

In order to model the deformation, a Mass Spring model combined with a Tensegrity model were used (Bourne 2007)

3.3 Ribs rotation

3.3.1 Mesh

In order to have X3D models for each rib, pre-existing rib meshes were converted into X3D files from MAYA. A typical rib is defined by around 200 different points.

Figure 3.3: *Mesh of one rib used for the project*

3.3.2 Model

As explained in Chapter 2.1.2, the rotation of the ribs can be defined by the combination of two different rotations: the 'bucket handle' and the 'pump handle' movement. Wilson et al (2001) measured the 'bucket handle' angle and the 'pump handle' angle for five subjects at FRC and TLC. These angles were used for this project. The rotation of the ribs was thus decomposed into two distinct rotations. We consider here that the rest position is at FRC. Thus at FRC, the rib does not have any rotation component.

Hence, the 'pump handle' rotation is defined as a rotation around the y axis with an angle varying from 0° at FRC to $\alpha_{TLC} - \alpha_{FRC}$ at TLC. The 'bucket handle' rotation is defined as a rotation around the x axis with an angle varying from 0° at FRC to at $\beta_{TLC} - \beta_{FRC}$ TLC. Values of these angles are shown in Table 3.1. The model used for the respiration pattern is a sinusoid.

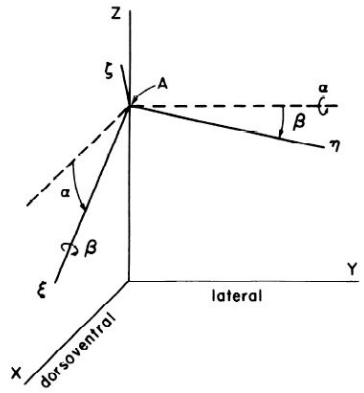


Figure 3.4*: The different rotation axis for the ribs*

Table 3-1: *Amplitude of the change in* α *and* β*. Data obtained from five subjects*

Rib no.	$\alpha_{TLC} - \alpha_{FRC}$ (deg)	$\beta_{TLC} - \beta_{FRC}$ (deg)
2	14.3	13.7
3	11.4	13.3
4	10.7	10.1
5	9.6	8.9
6	9.4	6.9
7	7.9	6.6
8	7.9	6.2
9	6	6.3

The rotation angles are given by the following equations:

$$\alpha' = (\alpha_{TLC} - \alpha_{FRC}) * \frac{1}{2} * (1 + \sin(2\pi ft))$$

$$\beta' = (\beta_{TLC} - \beta_{FRC}) * \frac{1}{2} * (1 + \sin(2\pi ft))$$

where:

α' is the reduced 'pump handle' angle
($\alpha'(FRC) = 0$ and $\alpha'(TLC) = \alpha_{TLC} - \alpha_{FRC}$)

β' is the reduced 'bucket handle' angle
($\beta'(FRC) = 0$ and $\beta'(TLC) = \beta_{TLC} - \beta_{FRC}$)

f is the respiration frequency

t is the time

3.3.3 Node implementation

Transform Node

The *Transform* Node is a grouping Node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. In particular, it contains a field called `rotation`, whose type is `SFRotation`. The `Rotation` type is defined by an arbitrary point and an axis. This `rotation` field has an effect on the object: it defines a geometric 3D rotation of the local coordinate system. Thus, if a *Transform* Node is created and if one of its children Node contains a rib model, it would be possible to apply a rotation to the rib.

The problem is that this field is not always updated. Setting an angle value and a rotation axis for this field just rotate the rib statically. The aim is to obtain a dynamic rotation.

TimeSensor Node

X3D specifications include a *TimeSensor* Node. This Node generates events as time passes. It can thus be used to drive continuous simulations and animations. One field is interesting: the field `time`. This field sends the absolute time for a given simulation tick (a simulation tick is defined by a scene-graph loop). These continuous events can be used to be routed to a field, which would be able to change the value of the `rotation` field in *Transform*.

RotTransform

It is necessary to create a new Node. The new Node is called *RotTransform* and inherits from *Transform.* Some new fields are created in this Node. A field `angles` is created. It is used to specify the 'bucket handle' and 'pump handle' angles. A TypedField `rota` is also created (see Chapter 3.1.4). The value type of this field is a `Rotation` type, but it has an `SFTime` input. Indeed, the `time` field is routed to `rota`. Thus, when an event is generated by the *TimeSensor* Node, `rota` is informed that its input field has changed. Since `rota` is also

routed to `rotation` - the field that rotates the local coordinate system – the value of `rotation` will change for each time step and will be equal to the value of `rota`.

Figure 3.5 summarizes the main fields, nodes and routes involved for the rotation of the ribs.

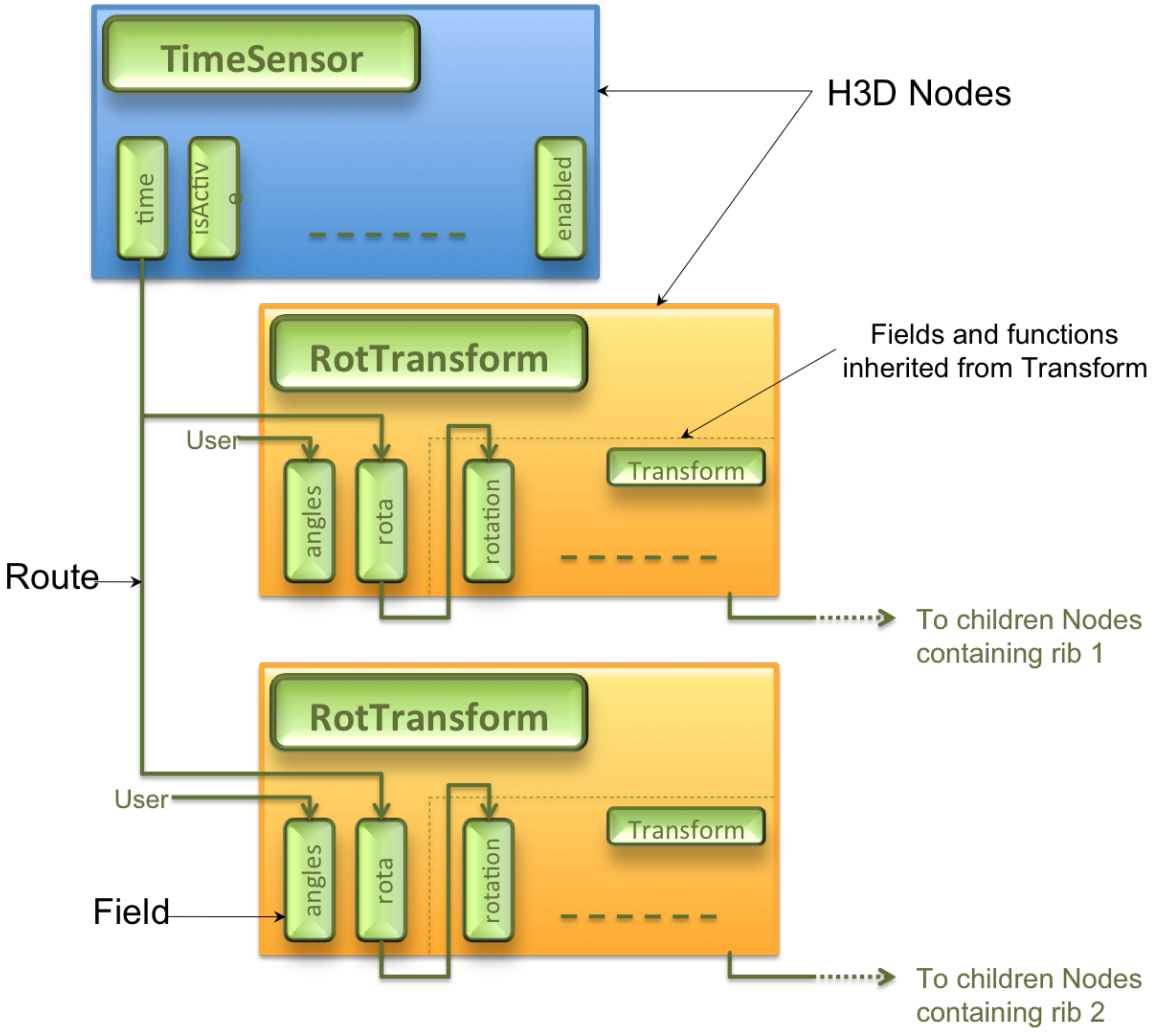


***Figure 3.5**: Overview of the RotTransform Node with its routes and fields applied with 2 ribs*

3.4 ChainMail deformer

3.4.1 Understanding the ChainMail model

When a ChainMail object is manipulated, it can stretch or contract itself according to strict displacement rules between a point and its neighbours. These rules are based on minimum and maximum distances that are allowed between a point and its neighbours. Thus the worst cases for a point and its neighbours are either maximal compression or maximal stretch (see Figure 3.6).

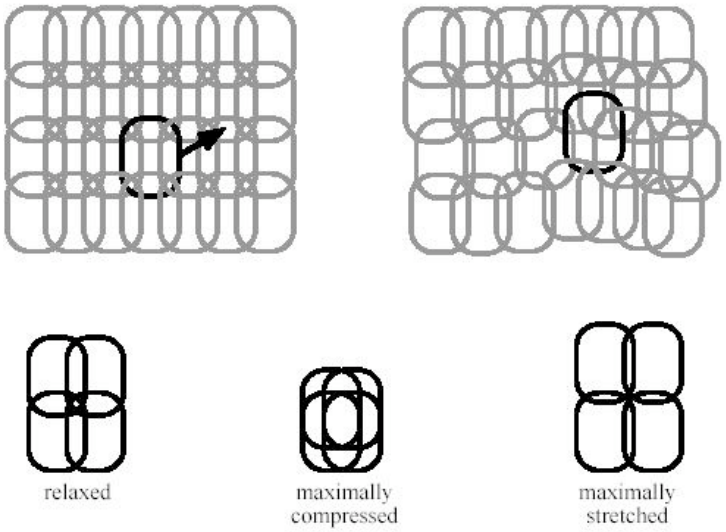


Figure 3.6: *Chainmail concept of compression and stretching*

Like the links in a chain, neighbours are forced to move only of the distances between the elements are violated. For example, a small displacement in a system where all links are stretched to their limit will cause the whole system to move. The parameters that control the allowable length of the links are modifiable. Thus, it is possible to model either rigid bodies or deformable objects.

3.4.2 Original Generalised ChainMail algorithm

The calculation of the deformation is based on the original algorithm proposed by Li and Brodlie (Li et Brodlie 2003). It is actually an extension of the original 3D Chainmail introduced by Gibson (Gibson 1997) to non-rectilinear grids. To understand the algorithm of the deformation for this project, it is important to first understand this original algorithm.

Data Structure

A ChainMail object contains the list of elements, the dimension of the object, compression factor, stretch factor, and shear factor. Each element contains its original position, its current position, a processing state, and a list of neighbours.

Position update

To understand how the position of an element is updated, let's consider element A, which is the one that is moved, and element B, which on of the neighbours of A. Element A is called the sponsoring element for B. Figure 3.7 shows the initial position of the elements. The rectangle represents the valid region for B. Figure 3.8 shows the final situation where A has moved, and B has followed.

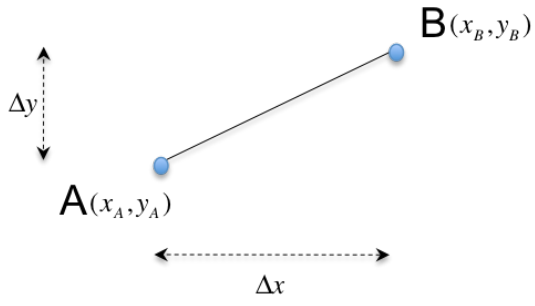


Figure 3.7: *original position of A and B*

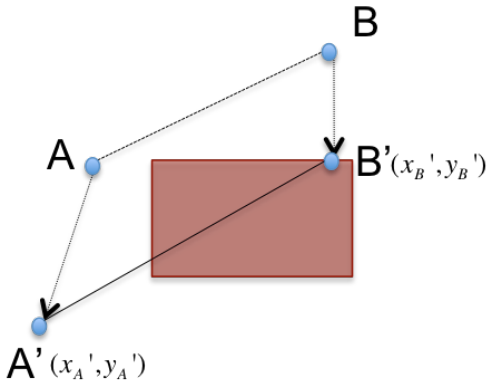


Figure 3.8: *A is moved to A' but B is not in the valid region anymore: it is moved to the nearest point of the valid region*

For the calculation, two distances are defined:

$$\Delta x = |x_A - x_B| \ ; \ \Delta y = |y_A - y_B|$$

Parameters that control the deformation behaviour are also defined. $\alpha_{\min}$ controls the compression. $\alpha_{\max}$ controls the stretching. β controls the shearing. It is then possible to define the boundaries of the valid region (the rectangle in Figure 3.8), supposing first that $x_B \geq x_A$ and $y_B \geq y_A$:

$$\begin{aligned} x_{\min} &= x_A' + (\alpha_{\min}\Delta x - \beta\Delta y) \\ x_{\max} &= x_A' + (\alpha_{\max}\Delta x + \beta\Delta y) \\ y_{\min} &= y_A' + (\alpha_{\min}\Delta y - \beta\Delta x) \\ y_{\max} &= y_A' + (\alpha_{\max}\Delta y - \beta\Delta x) \end{aligned}$$

where x_A' and y_A' are the new positions of the sponsoring element.

The valid region is then given by:

$$R = \{(x,y) : x_{\min} \leq x \leq x_{\max} ; y_{\min} \leq y \leq y_{\max}\}$$

It is important to notice that the softness and shearing parameters are expressed relative to the length of the original link rather than an absolute distance. This region is used during the calculation to check if the neighbours are still in a valid region.

The equation giving the boundaries changes if there is a different initial situation. For example, if $x_B \leq x_A$, the limits are given by:

$$\begin{aligned} x_{\min} &= x_A' - (\alpha_{\min}\Delta x - \beta\Delta y) \\ x_{\max} &= x_A' - (\alpha_{\max}\Delta x + \beta\Delta y) \end{aligned}$$

Deformation sequence

We now consider the whole system containing all the elements. There will be a cascade of element moves until each point is in a valid region. We suppose that element A is moved to a new position. Figure 3.9 shows how the algorithm works:

```
1    Update the position of element A
2    x_A = x_A'
3    y_A = y_A'
4    Add its neighbours to a waiting list
5    waiting_list.add( neighbours( A ) )
6    Process the waiting list
7    while ( waiting_list not empty ) {
8        element X = waiting_list.firstElement
9        if ( processing_state( X ) == true )
10           waiting_list.remove( X )
11       if ( processing_state( X ) == false ) {
12           Check if X is in the valid region with respect to its sponsor
13           if ( X is not in the valid region ) {
14               move X
15               waiting_list.add( neighbours( X ) )
16           }
17           processing_state( X ) = true
18       }
19   }
```

Figure 3.9: *Pseudo code of the Generalised ChainMail algorithm proposed by Li and Brodlie*

The algorithm continues as long as the waiting list is not empty. As each new element is added to the end of the waiting list, the deformation will spread out in a spiral from the element that is initially moved.

In this algorithm, each element is processed no more than once. The termination property is demonstrated in (Li et Brodlie 2003) in chapter 4.4.

3.4.3 Modelling respiration

In order to have a deformation of the object in this model, one initial element has to be moved. A deformation then propagates from this element and deforms the volume. However, to achieve a respiration simulation, it is not possible to move only one single point at the beginning of the deformation process. Thus this algorithm has to be adapted to the situation.

By modifying the algorithm, it is possible to calculate the deformation with multiple points that are moved simultaneously. We consider two elements A and B, which movement is forced simultaneously at the beginning of the process. The easiest way to calculate the induced deformation is first to calculate the deformation of the volume due to element A without considering the deformation of element B. The result is stored in a vector. Then the deformation due to element B is calculated without considering the one of element A and stored in a vector. Then for each element of the volume, the final deformation is obtained by averaging the position of the element when A was moved and the position of the element when B was moved.

Here, each element is processed no more than twice (the number of elements initially moved). Obviously, this represents a high computational cost so the number of initially moved points has to be optimized.

3.4.4 Implementation

DeformableShape

The *DeformableShape* Node is an X3DShapeNode, which makes it possible to deform the geometry of the shape when touching it with a Haptic device. The idea is to use the same structure than this Node to create a new Node that will make possible to deform the geometry of the shape when moving one or more point.

This new Node is called *DeformableShape2.* Like the first one, this Node contains interesting fields. The `deformer` field contains an H3DCoordinateDeformerNode that determines how the deformation should be done. The `origCoord` contains the coordinate of the object before any deformation started. The `deformedCoord` contains the coordinate after the deformation and that will be used for graphics rendering.

A C++ class has been created in order to find the neighbours of each element. The first time *DeformableShape2* is called, each element of the shape is initialized by finding all its neighbours and storing them in a *neighbourhood.*

In *DeformableShape* (the original Node dealing with Haptic deformation), there is a call to the function *`deformPoints`* that is implemented in another class in order to calculate the deformation. In this project, the deformation will be calculated in the function `deformPoints2`. This function will be a member of another new Node: *ChainmailDeformer.*

ChainmailDeformer

The main feature of the Node *ChainmailDeformer* is the function `deformPoints2`. The main inputs of the function are the initial coordinates of the object, the coordinates of the object at the previous time step, the whole neighbourhood of the object, the softness and shearing parameters, the index of the points that are voluntarily moved, and the new position of these points. The main output is the new deformed coordinates. The implementation of this function follows the algorithm described in Figure 3.9 and in 3.4.3

The Node also contains a function that checks if a neighbour is in the valid region in regards to its sponsoring element and in case, that moves this neighbour inside the valid region.

3.5 Diaphragm

The Figure 3.10 shows the mesh that is used for the diaphragm.

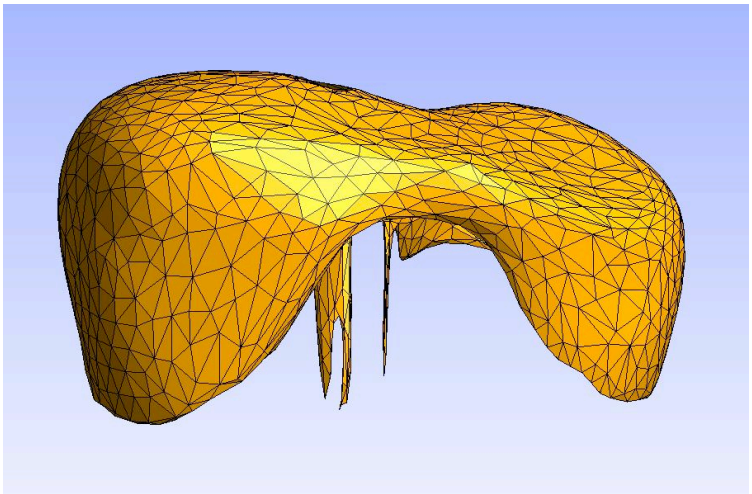


Figure 3.10: *The diaphragm mesh that is used for the project*

The movement of the diaphragm has two main features. The first movement is due to the contraction of the muscle and the rotation of the ribs. The diaphragm is contracted and it then looks smaller. Secondly, the central tendon has got a movement almost parallel with the transverse plane.

To model the central tendon as a tendon and not a muscle, it has to be almost rigid. This can be done by settings the ChainMail parameters to specific values. When the compression parameter is equal to *1*, the stretch parameter is equal to *1* and the shear parameter is equal to *0*, the object is rigid.

The diaphragm cannot be modelled as a homogeneous volume. The vector `rigidity` has thus been created. This vector gives to each point of the volume specific ChainMail parameters (compression, stretch, shear). This way, the rigidity values of the points shown in red in Figure 3.11 have been set *to (1, 1, 0)*, and the central tendon is rigid.

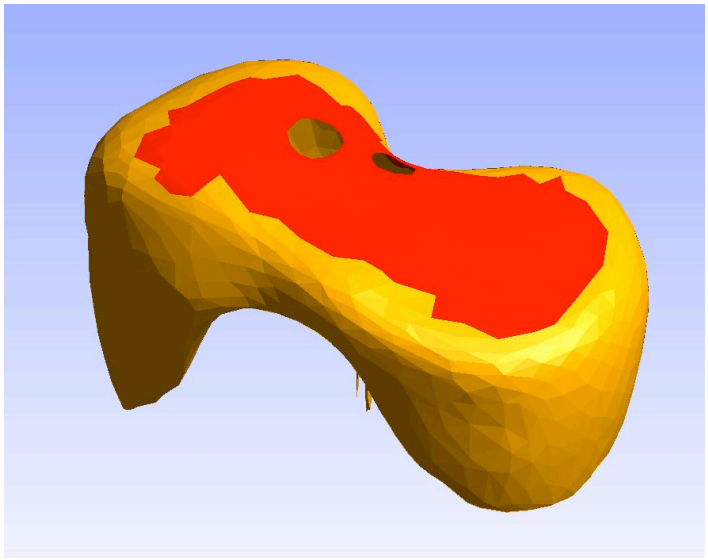


Figure 3.11: *The red elements in the diaphragm are rigid*

In the *DeformableShape2* Node, a new field `rigid_points` has been created to put the points that are going to be rigid in.

The movement of this central tendon is nearly vertical (see Chapter 2.1.1). To do this, one central point is forced to have a sinusoidal movement along the vertical axis. During inhalation, the central tendon has a downward movement, which is synchronous with the expanding rib cage. As the links for the central tendon are rigid, the points corresponding to this tendon will follow the exact same movement. The other points will have the expected "chain reaction" according to the ChainMail rules.

At FRC, the diaphragm has a shape like the one that can be seen in Figure 3.10. Thus it must be able to compress itself during inhalation. The chosen value for the compression parameter is 0.7. The stretch and shear components are not very important here. The value for the stretch parameter and the shear parameter are 1.1 and 0.1.

However, the diaphragm is also supposed to follow the movement of the ribs. To do this, the idea is to 'fix' some points of the diaphragm to some points of the ribs.

Coordinates of the moving ribs

As explained in Chapter 3.3.3, the action of the *RotTransform* Node is to do a rotation of the local coordinate system for each rib. The Node contains a field called matrix, which is the transformation matrix for the object (rotations, translations, etc). In order to have access to the coordinates of the ribs in real time, the original coordinates have to be multiplied by this matrix. The equation of the transformation is given below:

$$P' = M.P$$

where P are the original coordinates and P' the coordinates of the moving rib.

These coordinates are put in a field so that it can be routed to another field.

Calculation of the contact points

The *RotTransform* Node is also used to calculate which are the best contact points between two objects. One Node has to be created for each contact between two different objects. Two input fields `coord1` and `coord2` receive the coordinates of the two contact objects – object 1 and object 2. Another field called `distance` has been created so that the user can define how the two objects should be linked.

The first value in `distance` is a maximum distance between two points of the objects. If the distance between two points - A from object 1 and B from object 2 - is lower than this maximum distance, B will be set to follow A.

The second value in `distance` is also a maximum distance between two points. If the distance between A and B is lower than this maximum distance, the ChainMail parameters of B will be set to be rigid. For example, if there is one contact point between one rib and the diaphragm, it will be more realistic if the region near the contact point is rigid. Furthermore, this allows for less contact points and thus less computational time.

The three last values in `distance` are boundaries. The third and fourth values are limits in the dorso ventral direction. The calculation of distances will be done only within these limits. The last value is a minimum limit in the left-right direction. These boundaries allows for more precision in the choice of contact points.

Once the calculation has been done, one field `touch` is filled with the coordinates of the points in object 1 that have to be followed and the points in object 2 that have to be set to follow points in object 1. Another field `rigid_touch` is filled with points which rigidity has to be set to rigid.

Movement of the diaphragm

The movement of the diaphragm will be the combination of the downward movement of the central tendon and the movement induced by the contact points with the ribs.

3.6 Lungs

The model used for the lungs in the project can be seen in Figure 3.12.

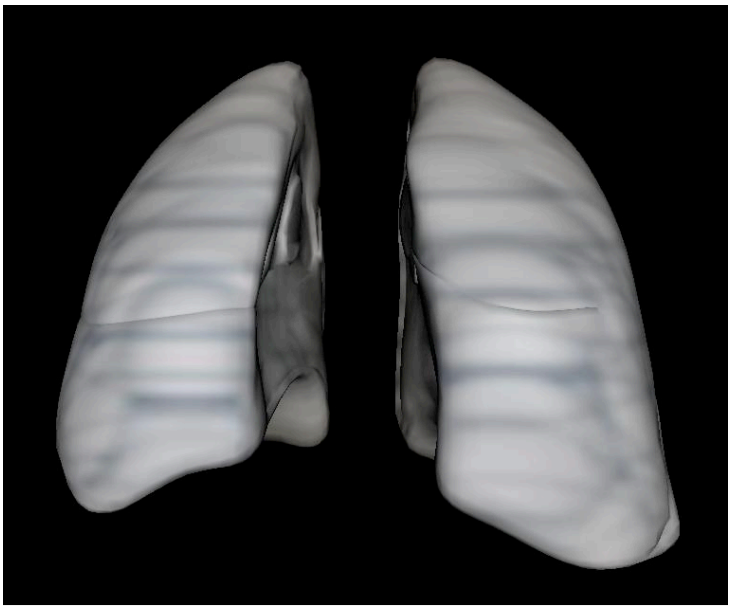


Figure 3.12: *Lung model used in the project*

The lungs have to move with the diaphragm and at the same time with the ribs. The calculation of contact points between the ribs and the lungs, and between the diaphragm and

the lungs is done in the *RotTransform* Node (see Chapter 3.6). Extremities (at the top and the bottom) have to be set to rigid in order to have a more realistic movement.

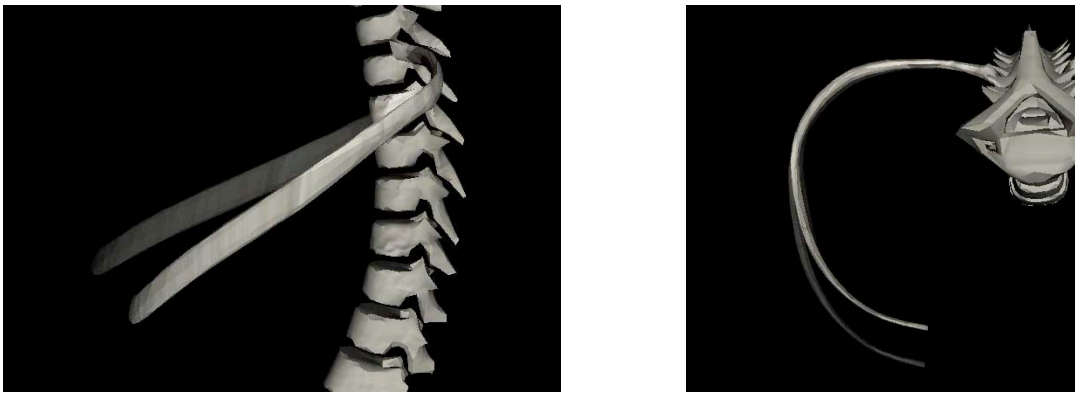
4 Results

This chapter presents the results of the simulation.

4.1 Ribs

4.1.1 Pump handle

The rotation or the ribs is decomposed in a 'pump handle' movement and a 'bucket handle' movement. Figure 4.1 only shows the 'pump handle' movement for one rib. The rotation is around the left-right axis.

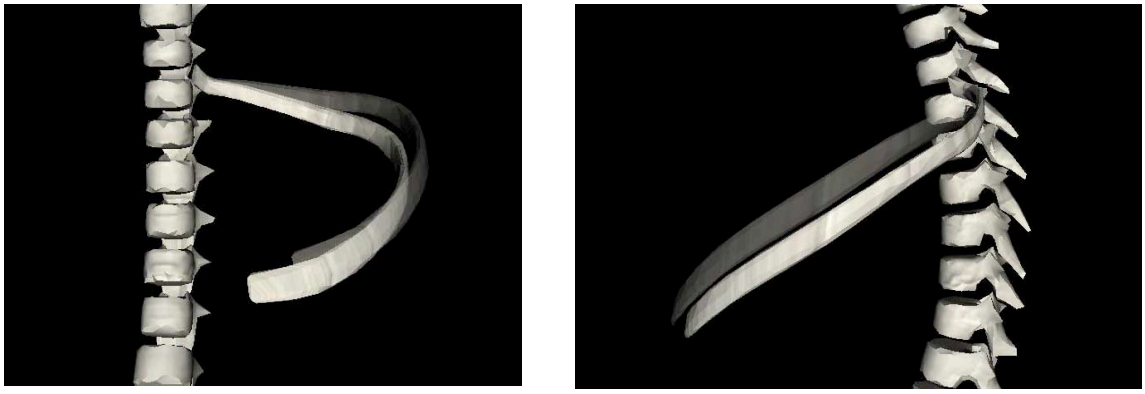


(a) (b)

Figure 4.1: *Pump handle movement of the 5th rib.*
In bright: 5th rib at FRC. In dark: 5th rib at TLC. (a) View from the left side
(b) View from the top of the right rib

4.1.2 Bucket handle

Figure 4.2 only shows the 'bucket handle' movement. The rotation is along the dorso ventral axis.

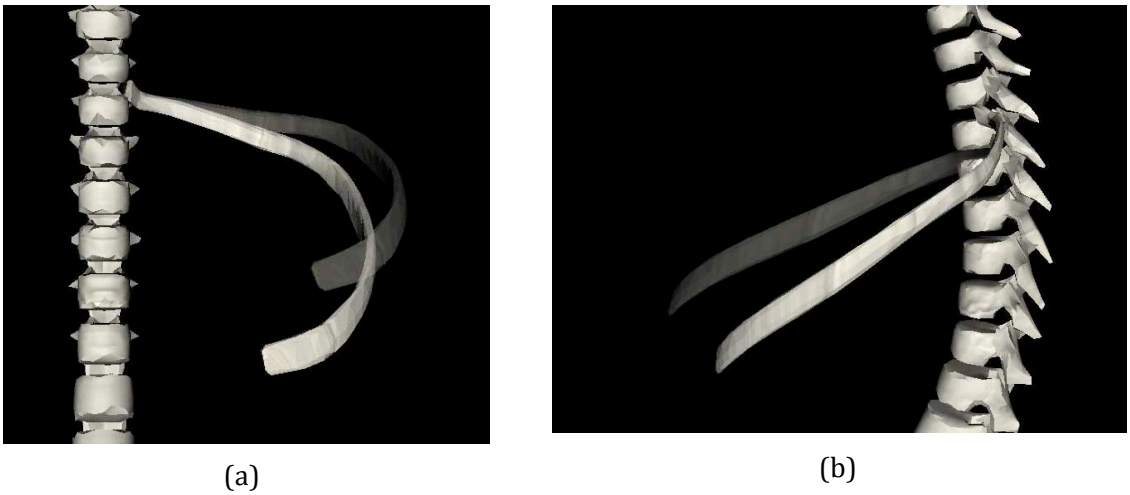


(a) (b)

Figure 4.2: *Bucket handle movement of the 5th rib.*
In bright: 5th rib at FRC. In dark: 5th rib at TLC. (a) View from the front
(b) View from the left side

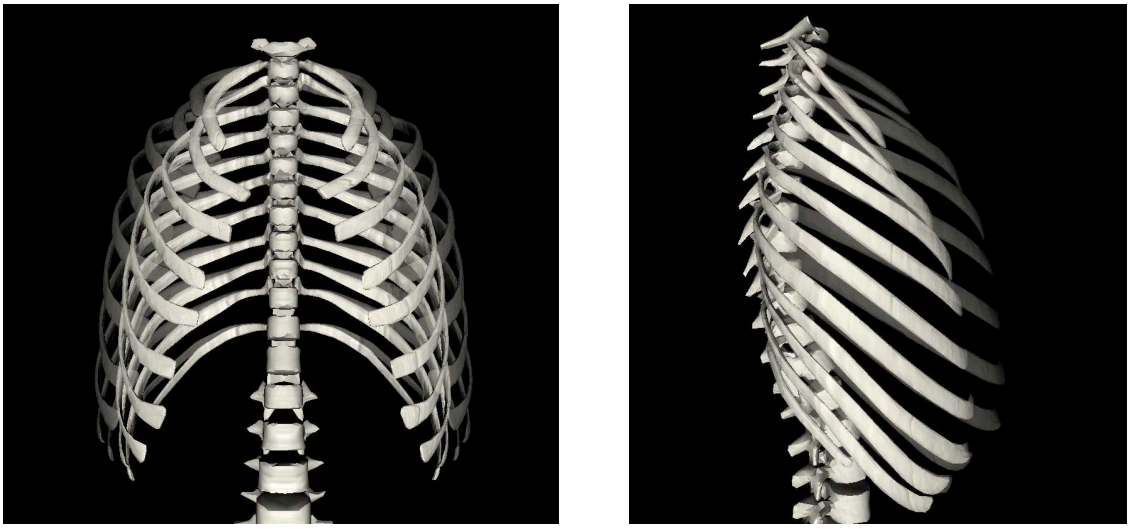
4.1.3 Total

Figure 4.3 shows the complete movement for the rotation of the 5th rib.



(a) (b)

Figure 4.3: *Complete Rotation of the 5th rib.*
In bright: 5th rib at FRC. In dark: 5th rib at TLC. (a) View from the front
(b) View from the left side



(a) (b)

Figure 4.4: *Rotation of the ribs at FRC (bright) and TLC (dark)*

Figure 4.4 shows that during inhalation, the volume of the rib cage increases in both the anteroposterior and the transverse direction.

4.2 Diaphragm

To obtain a realistic movement of the diaphragm, two contact points for each side have been defined. The diaphragm is thus in contact with the 10th and the 11th ribs. Table 4.1 shows the values that have been found for the calculation of the contact points in order to have a realistic behaviour. There are thus 4 constrained points due to the movement of the ribs (1 for each rib).

Furthermore, the central tendon is also moving. This is due to the movement of one central point. There are finally 5 constrained points for the diaphragm: 2 on each side and 1 at the top.

Figure 4.5 shows the movement of the diaphragm during the simulation.

Table 4-1: *Values of the* `distance` *field in RotTransform Nodes (Diaphragm)*

	Minimum distance for the contact points	Minimum distance for the rigidity	z_{min} (dorso ventral axis)
Left Rib 10 & Diaphragm	0.55	1.5	-0.5
Right Rib 10 & Diaphragm	0.58	1.5	-0.5
Left Rib 11 & Diaphragm	0.51	1.5	-5.8
Right Rib 11 & Diaphragm	0.54	1.5	-5.8

Table 4.1 only shows the 3 first values of the `distance` field since the others were not relevant.

As expected, the diaphragm looks smaller at TLC.

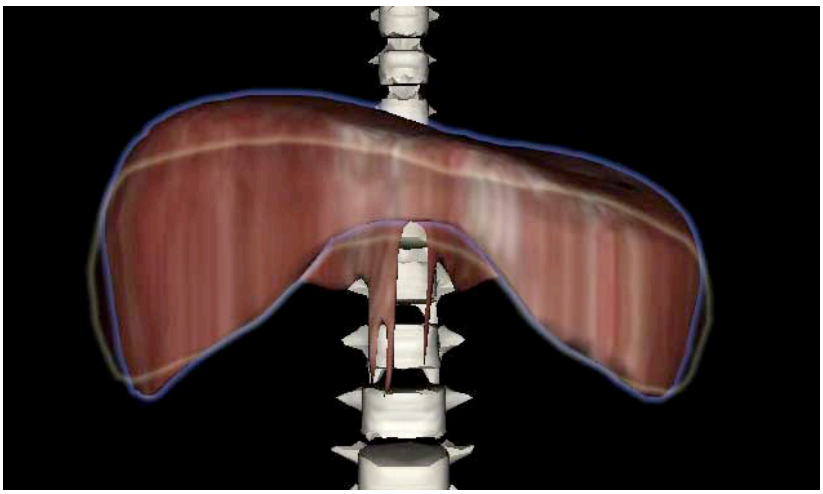


Figure 4.5: *Shape of the diaphragm at FRC (blue line) and TLC (yellow line)*

4.3 Lungs

In order to minimize the number of contact points between the lungs and the ribs, only two ribs are going to induce the movement. The 2nd rib and the 5th rib showed the most

realistic results. Table 4.2 shows the values that have been found for the calculation of the contact points between the lungs and the ribs.

Table 4-2: *Values of the* `distance` *field in RotTransform Nodes (Lungs)*

	Minimum distance for the contact points	Minimum distance for the rigidity	z_{min} (dorso ventral axis)
Left Rib 2 & Left Lung	0.48	3	0
Left Rib 5 & Left Lung	0.22	2	0
Right Rib 2 & Right Lung	0.145	3	0
Right Rib 5 & Right Lung	0.13	2	0

There is a limit in the dorso ventral axis (see Table 4.2 column 3) otherwise the contact point would be situated close to the spine, which is not giving realistic results. There is 1 contact point for each rib. Thus there are 2 contact points between the ribs and each lung.

Table 4.3 shows the values that have been found for the calculation of the contact points between the lungs and the diaphragm. Once again, only 1 contact point for each lung has been chosen.

Table 4-3: *Values of the* `distance` *field in RotTransform Nodes (Lungs)*

	Minimum distance for the contact points	Minimum distance for the rigidity	z_{min} (dorso ventral axis)	z_{max} (dorso ventral axis)	x_{min} (left right axis)
Diaphragm & Left Lung	0.19	2	-2	2	-20
Diaphragm & Right Lung	0.65	2.5	-2	2	-6

Eventually, each lung has got 3 constrained points. Figure 4.6 shows the movement of the lungs during respiration with 3 constrained points per lung.

Figure 4.6: *Shape of the lungs at FRC (yellow line) and TLC (red line)*

4.4 Complete model

The complete model contains the ribs, the spine, the sternum, the diaphragm, the lungs and the cartilages between the sternum and the ribs. Ribs are rotating according to the movement described in 4.1. The diaphragm and the lungs are moving according to the movements described in 4.2 and 4.3. The sternum is rotating around the left-right axis – it is as if there was only a 'pump handle' movement. The cartilages are rotating with the sternum, but they are not attached to the ribs and are not deforming.

This complete model has been integrated into an environment including a Haptic interface (the interface has been developed in a parallel project). It is possible to navigate around the model via the Haptic device, to rotate around the model via buttons on the Haptic device. The stereoscopic display is enabled – it is thus much easier to manipulate the Haptic stylus. It is also possible to touch and feel the objects of the model.

The frame rate is 2.5 for the graphics, and 1024 for the Haptics. A 'pause' and 'resume' button has also been created. The 'pause' button stops the respiration. This allows for a much smoother navigation. Figure 4.7 shows the complete model.

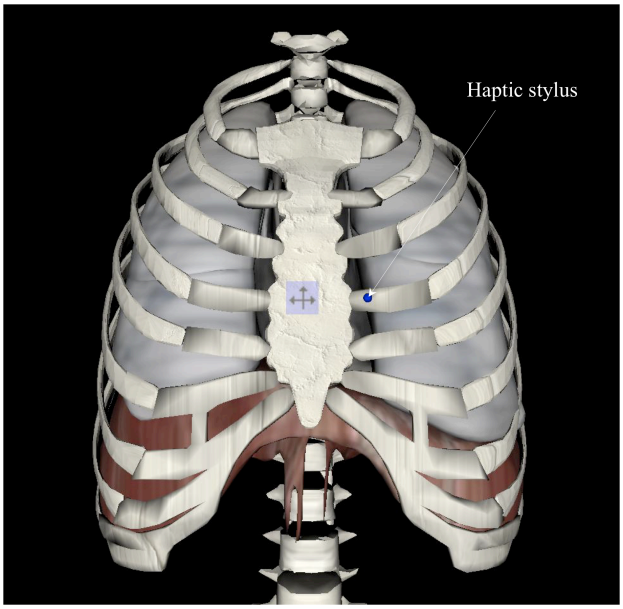


Figure 4.7: *Complete model with Haptic stylus*

4.5 Evaluation

4.5.1 Protocol

In order to have an early feedback on the model and assess its feasibility, it had to be tested by teachers and students. Thus students received a 30 minutes small group teaching session aided by a display where the deformable model is projected in 3D stereo, a Haptic device and 2D diagrams and illustrations from textbooks. The session dealt with an introduction to lung function. Upon completion, students and teachers filled in survey questionnaires and received semi-structured interviews. Two video cameras were recording the teaching session.

Table 8.1 shows the timetable of the session.

4.5.2 Results

In Table 4.4 and 4.5, the red figures represent the choice of the teacher, and the black figures the choice of the student. Only one teacher and one student took part to this first session.

In addition to this questionnaire, the student and the teacher were interviewed and gave a feedback on their experience.

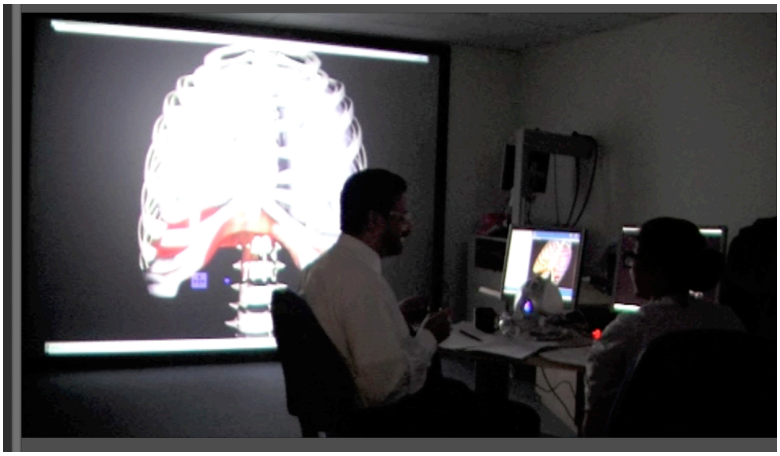


Figure 4.8: *Teaching session on the lung physiology with help of the dynamic model*

Table 4-4: *Results of the questionnaire: Rating of technology*

	Very good	Good	Adequate	Bad	Very Bad
Overall usability		1	**1**		
Overall quality		**1**	1		
Haptic device		1	**1**		
Stereo display	1	**1**			
3D Model		**1**	1		

Table 4-5: *Results of the teaching questionnaire: Rating of the usefulness of the technology for learning*

	Very high	High	Adequate	Low	Very Low
To supplement lectures		1 + 1			
To supplement textbooks		1 + 1			
To supplement small group tutorials	1 + 1				
For initial learning			1 + 1		
For revision	1 + 1				

5 Discussion

The overall feedback given by the student and the teacher were quite good. They tested the simulation at an early stage of the development, so they tried to guide us for the future.

5.1 Model

Ribs rotation

The rotation of the ribs is successful. The data collected is highly valuable and allowed for precise rotation angles. Moreover it is still possible and easy to adjust these angles by changing the values of a field in a X3D file. Finally, the computational cost is not too high for the ribs rotation alone.

Diaphragm and lungs deformation

The deformation of the diaphragm is quite realistic. It is a good surprise given that there are only 5 constrained points.

The deformation of the lungs was however less realistic. It is harder to model the lung movement with only 3 constrained points. For example it does not always look synchronous with the ribs. The trade-off between rigidity (to be able to make the lungs move with the ribs and diaphragm) and elasticity (to allow for inflation) is hard to find.

Speed

One of the biggest problems so far is the speed of calculation. It has clearly been improved since the initial structure has been changed: the biggest calculations are now done in the initialisation (find the neighbours, calculate the rigidity, calculate the contact points,

calculate the number of constrained points). However, the computational cost at each step is still too high. The algorithm used to calculate the deformation with multiple points initially moved is still not completely optimized. Restricting the number of constrained points is not sufficient enough to overcome this problem.

Complete model

The stereoscopic display and Haptics rendering are really good when the model is not moving. As the speed of calculation is still too slow, it is hard to navigate with Haptic device while the organs are moving.

It has also been noticed that the position of the ribs relative to the lungs is not very realistic. The gap between them looks too big.

5.2 Usefulness for teaching

The potential of such an application is really big. It seems to really help students understand the function and structure of the rib cage. The movement of the ribs, the diaphragm and the lungs can be seen simultaneously. Both teacher and student found it useful for the teaching session. Even if you cannot replace the initial learning, this kind of application seems really helpful when it comes to practicals or revision. The possibility to play with it, to touch, to feel, to pause or to rotate is really stimulating.

Given that it was a first version of the application and that a lot of missing features could be added, both teacher and student were confident in the future of this model.

5.3 Future work

It would be interesting to have the possibility to label the organs that are touched, and to add or remove some of them in real time. This can be done thanks to an environment developed in a parallel project.

It is possible to touch the organs, but there is no difference in the feeling between the lungs and the diaphragm for example. It would be interesting to differentiate two different organs just by the touch sensation and the Haptic feedback.

It is important to see the muscles that cause respiration. Diaphragm is one of the most important, but it would be interesting to add other muscles that are actually as important as the diaphragm.

The model could obviously be extended by adding other structures like the liver, the guts, the heart or even higher with the sternocleidomastoid muscle.

Finally, the model will be much more pleasant and useful if it has a proper interface. It should be possible to change parameters (like a rotation angle) easily via an interface. The interface could include several operation modes. Thus it would be useful to have different respiration patterns. Indeed the muscles involved in quiet breathing are not the same as those involved in augmented breathing.

To conclude, it seems crucial to continue the development of such applications. The future users see in it a very powerful tool. Seeing the anatomy in three dimensions and moving it is helpful because it is important to relate the function to the structure. That is what the physiology is about.

6 Works Cited

Arora, NS, and DF Rochester. "Effect of body weight and muscularity on human diaphragm muscle mass, thickness, and area." *J. Appl Physiol* 52 (1982): 64-70.

Barrows, HS, PR Patek, and S Abrahamson. "Introduction of the living human body in freshman gross anatomy." *Br J Med Educ* 2 (1968): 33-35.

Bourne, Wesley. "The Use of Tensegrity to Simulate Diaphragm Motion Through Muscle and Rib Kinematics." MSc Thesis, Department of Computing, Imperial College London, 2007.

Cluzel, Philippe, Thomas Similowski, Carl Chartrand-Lefebvre, Marc Zelter, Jean-Philppe Derenne, and Philippe A. Grenier. "Diaphragm and ChestWall: Assessment of the Inspiratory Pump with MR Imaging— Preliminary Observations1." *Radiology* 215, no. 2 (2000): 574-583.

Didier, Anne-Laure, Pierre-Frédéric Villard, Jean-Yves Bayle, Michaël Beuve, and Behzad Shariat. "Breathing Thorax Simulation based on Pleura Physiology and Rib Kinematics." *Medical Information Visualisation - BioMedical Visualisation, 2007. MediVis 2007. International Conference on.* 2007. 35-42.

Dräger, Christopher. *A ChainMail Algorithm for Direct Volume Deformation in Virtual Endoscopy Applications.* Master Thesis, Vienna University of Technology, Vienna: VRVis Research Center, 2005.

Gibson, Sarah F. F. "3D ChainMail: a Fast Algorithm for Deforming Volumetric Objects." *Symposium on interactive 3D Graphics.* 1997. 149 - 154.

Gray, Henri. *Anatomy of the Human Body.* 1918.

H3D.org - Open Source Haptics. http://www.h3dapi.org (accessed September 2008).

Haluck, Randy, Roger Webster, Dean Zimmerman, Betty Mohler, Alan Synder, and Mike Melkonian. "A Prototype Haptic Suturing Simulator." *Stud Health Technol Inform* 81 (2001): 567-569.

Kaye, Jonathan, Dimitris N. Metaxas, and Frank P. Primiano. "A 3D virtual environment for modeling mechanical cardiopulmonary interactions."

Li, Ying, and Ken Brodlie. "Soft Object Modelling with Generalised ChainMail - Extending the boundaries of Web-based Graphics." *Computer Graphics* 22, no. 4 (2003): 717-727.

Madani, H., P.A. Paraskeva, and A. Darzi. "A minimally invasive approach to undergraduate anatomy teaching." *Anatomical Sciences Education* 1 (2008): 46-47.

McKachlan, John C, and Debra Patten. "Anatomy teaching: ghosts of the past, present and future." *Medical Education* 40 (2006): 243-253.

Meier, U., O. Lòpez, C. Monserrat, M.C. Juan, and M. Alcañiz. "Real-time deformable models for surgery simulation: a survey." *Computer Methods and Programs in Biomedicine* 77 (2005): 183-197.

Paalman, Mark H. "Why teach anatomy ? Anatomists respond." *The Anatomical Record* 261 (2000): 1-2.

Patel, KM, and BJ Moxham. "Attitudes of professional anatomists to curricular change." *Clin Anat* 19 (2006): 132-141.

Pickering, Mark, and James F. X. Jones. "The diaphragm: two physiological muscles in one." *J.Anat*, no. 201 (2002): 305-312.

Raftery, Andrew T. "Anatomy teaching in the UK." *Surgery* 25, no. 1 (2006).

Schill, Markus A., Sarah F. Frisken Gibson, H.-J. Bender, and R. Manner. "Biomechanical Simulation of the Vitreous Humor in the Eye Using an Enhanced ChainMail Algorithm." *Proceedings of Medical Image Computation and Computer Assisted Interventions.* 1998. 679-687.

Terzopoulos, Demetri, John Platt, Alan Barr, and Kurt Fleischer. "Elastically Deforable Models." *Computer Graphics* 21, no. 4 (1987): 205-214.

The Thoracic Cage/Respiration & Breathing. http://www.courses.vcu.edu/DANC291-003/unit_4.htm (accessed September 2008).

Wilson, Theodore A., Alexandre Legrand, Pierre-Alain Gevenois, and Andrãe De Troyer. "Respiratory effects of the external and internal intercostal muscles in humans." *Journal of Phyiology* 530, no. 2 (2001): 319-330.

7 Appendices

7.1 Test session

Table 7-1: *Timetable of the teaching test session*

Time	Event	Location	Action
14.00 9th Sept			Set up & test hardware and software
12.30 10th Sept	MJ arrives		Set up for session 2 with HB
1.00-1.30	SM arrives		Briefing with HB, training
14.20	Participants arrive		Briefing with HB and SM Sign consent form
14.30	Session 2 starts	Demo room	
15.05	Participant group interview	Demo room	
15.30	Teacher interview	1035	Teacher interview HB
15.35	Download files & cleanup	Demo room 1035	

Contacts:	MJ	Mathieu Jacob	Researcher
	SM	Lung teacher	
	HB	Harry Brenton	Researcher

7.2 ChainmailDeformer Node

7.2.1 ChainmailDeformer.h

```
#ifndef __ChainmailDeformer_H__
#define __ChainmailDeformer_H__

#include <queue>
#include "SFFloat.h"
#include "DeviceInfo.h"
#include "H3DCoordinateDeformerNode2.h"

#include <iostream>
#include <string>
#include <sstream>

namespace H3D {
  /// \ingroup H3DNodes
  /// The ChainmailDeformer determines a deformation according to Generalised
  /// Chainmail algorithm from Ken BRODLIE and Ying LI
  class  ChainmailDeformer : public H3DCoordinateDeformerNode2 {
  protected:
     /// Compute area inside which the neighbour is allowed to lie.
     /// This function moves the neighbour inside its allowed area if it is outside.
     bool placeNeighbourInItsArea ( const Vec3f& org_coord, const Vec3f& coord, const
Vec3f& org_neighb, Vec3f& neighb, Vec3f rigid );
  public:
     /// Constructor
     ChainmailDeformer( Inst< SFFloat > _compressionFactor =0 ,
                                Inst< SFFloat > _stretchFactor     =0 ,
                                Inst< SFFloat > _shearFactor       =0 ):
       compressionFactor( _compressionFactor ),
       stretchFactor( _stretchFactor ),
       shearFactor( _shearFactor )
       {

       type_name = "ChainmailDeformer";
       database.initFields( this );

       compressionFactor->setValue( 0.33333 );
       stretchFactor->setValue( 1.25 );
       shearFactor->setValue( 0.25 );
     }

   virtual void deformPoints2(

                                                 const vector<  int > idx_point,

                              const vector< Vec3f > &penetration_points,
                                          const vector< Vec3f > &rigidity,

                                          const vector< vector< unsigned long > >
neighb_total,
```

```
                                const vector< Vec3f > &orig_points,

                                const vector< Vec3f > &deformed__points,

                                vector< Vec3f > &new_deformed_points );


    /// This factor is used with original lengths to compute
    /// minimum allowed length between a point and its neighbour(s).
    ///
    /// <b> Access type: </b> inputOutput
    auto_ptr< SFFloat > compressionFactor ;

    /// This factor is used with original lengths to compute
    /// maximum allowed length between a point and its neighbour(s).
    ///
    /// <b> Access type: </b> inputOutput
    auto_ptr< SFFloat > stretchFactor ;

    /// This factor is used with original lengths to compute
    /// allowed interval for others coordinates.
    /// For instance, if stretchFactor and compressionFactor are used to cumpute
    /// x allowed length, shearFactor is used for y and z allowed length.
    ///
    /// <b> Access type: </b> inputOutput
    auto_ptr< SFFloat > shearFactor ;

    /// The H3DNodeDatabase for this node.
    static H3DNodeDatabase database;
  };
}

#endif
```

7.2.2 ChainmailDeformer.cpp

```
#include "ChainmailDeformer.h"

using namespace H3D;

// Add this node to the H3DNodeDatabase system.
H3DNodeDatabase ChainmailDeformer::database( "ChainmailDeformer",
                                    &(newInstance<ChainmailDeformer>),
                                    typeid( ChainmailDeformer ),
                                    &H3DCoordinateDeformerNode2::database );

namespace ChainmailDeformerInternals {
     FIELDDB_ELEMENT( ChainmailDeformer, compressionFactor, INPUT_OUTPUT );
     FIELDDB_ELEMENT( ChainmailDeformer, stretchFactor, INPUT_OUTPUT );
     FIELDDB_ELEMENT( ChainmailDeformer, shearFactor, INPUT_OUTPUT );
}

bool ChainmailDeformer::placeNeighbourInItsArea ( const Vec3f& org_coord, const Vec3f&
coord, const Vec3f& org_neighb, Vec3f& neighb, Vec3f rigid )
{    bool      res( false );
```

```
    float     deltaX( abs( org_coord.x-org_neighb.x ) ),
              deltaY( abs( org_coord.y-org_neighb.y ) ),
              deltaZ( abs( org_coord.z-org_neighb.z ) ),
              minX, maxX,
              minY, maxY,
              minZ, maxZ;

    float     compression( rigid.x ),
              stretch( rigid.y ),
              shear( rigid.z );


if ( org_neighb.x >= org_coord.x ) {
       minX = coord.x + compression * deltaX - shear * ( deltaY + deltaZ );
       maxX = coord.x + stretch * deltaX + shear * ( deltaY + deltaZ );
    } else {
       minX = coord.x - compression * deltaX + shear * ( deltaY + deltaZ );
       maxX = coord.x - stretch * deltaX - shear * ( deltaY + deltaZ );
    }
    if ( org_neighb.y >= org_coord.y ) {
       minY = coord.y + compression * deltaY - shear * ( deltaX + deltaZ );
       maxY = coord.y + stretch * deltaY + shear * ( deltaX + deltaZ );
    } else {
       minY = coord.y - compression * deltaY + shear * ( deltaX + deltaZ );
       maxY = coord.y - stretch * deltaY - shear * ( deltaX + deltaZ );
    }
    if ( org_neighb.z >= org_coord.z ) {
       minZ = coord.z + compression * deltaZ - shear * ( deltaX + deltaY );
       maxZ = coord.z + stretch * deltaZ + shear * ( deltaX + deltaY );
    } else {
       minZ = coord.z - compression * deltaZ + shear * ( deltaX + deltaY );
       maxZ = coord.z - stretch * deltaZ - shear * ( deltaX + deltaY );
    }


    if (minX>=maxX)
    {
       if ( neighb.x > minX ) {
              neighb.x = minX;
              res = true;
              }
       else if ( neighb.x < maxX ) {
              neighb.x = maxX;
              res = true;
              }
    }
    else
    {
       if ( neighb.x < minX ) {
              neighb.x = minX;
              res = true;
       }
       else if ( neighb.x > maxX ) {
              neighb.x = maxX;
              res = true;
       }
```

```
    }


    if (minY>=maxY)
    {
      if ( neighb.y > minY ) {
              neighb.y = minY;
              res = true;
      }
      else if ( neighb.y < maxY ) {
              neighb.y = maxY;
              res = true;
      }
    }
    else
    {
      if ( neighb.y < minY ) {
              neighb.y = minY;
              res = true;
      }
      else if ( neighb.y > maxY ) {
              neighb.y = maxY;
              res = true;
      }
    }

    if (minZ>=maxZ)
    {

      if ( neighb.z > minZ ) {
              neighb.z = minZ;
              res = true;
      }
      else if ( neighb.z < maxZ ) {
              neighb.z = maxZ;
              res = true;
      }
    }
    else
    {
              if ( neighb.z < minZ ) {
              neighb.z = minZ;
              res = true;
      }
      else if ( neighb.z > maxZ ) {
              neighb.z = maxZ;
              res = true;
      }
    }

    return res;



}
```

```
void ChainmailDeformer::deformPoints2(
                                                        const vector<  int >
idx_point,
                                    const vector< Vec3f > &penetration,
                                                        const vector< Vec3f >
&rigidity,

                                                    const vector< vector<
unsigned long > > neighb_total,
                                                    const vector< Vec3f >
&orig_points,

                                const vector< Vec3f > &deformed_points,

                                                    vector< Vec3f >
&new_deformed_points ) {
```

```
        new_deformed_points = deformed_points;
        unsigned int nr_devices = penetration.size();

        if( nr_devices > 0 ) {
                        vector< unsigned int > index_point_to_move;
                        vector< unsigned int > touch_point_to_update;

                        index_point_to_move.assign( idx_point.begin(), idx_point.end() );


                        vector< unsigned long > neighbours;
                        Vec3f   coord, org_coord,
                                        neighb, org_neighb;
                        unsigned long coord_idx;
                        vector< bool > processing_state( orig_points.size() );

                        fill( processing_state.begin(), processing_state.end(), false );
                        stack< unsigned long > moved_elements;
                        queue< pair< unsigned long, vector< unsigned long > > >
candidate_for_move;

                        vector<vector< bool > > processing_state_vector(nr_devices,
processing_state);
                        vector<vector< Vec3f > >
new_deformed_vector(nr_devices,new_deformed_points);

                        for ( unsigned int j(0); j!=1; j++ )
                                {

                                new_deformed_vector[ j ][ index_point_to_move[ j ] ] =
penetration[ j ] ;
                                processing_state_vector[ j ][ index_point_to_move[ j ] ] =
true;
```

```
                                moved_elements.push( index_point_to_move[ j ] );
                                neighbours = neighb_total[ index_point_to_move[ j ] ];

                                // add moved element neighbours to the candidates for moving
list
                                candidate_for_move.push( pair<unsigned long, vector<
unsigned long > >( index_point_to_move[ j ], neighbours ) );


                                while( candidate_for_move.size() )
                                {

                                                coord_idx = candidate_for_move.front().first;
                                                neighbours =
candidate_for_move.front().second;
                                                coord = new_deformed_vector[ j ][ coord_idx
];
                                                org_coord = orig_points[ coord_idx ];

                                                for( unsigned int i(0), I(neighbours.size());
i!=I; i++ )
                                                {
                                                        if ( processing_state_vector[ j ][
neighbours[ i ] ] == false )
                                                        {

                                                                neighb = new_deformed_vector[
j ][ neighbours[ i ] ];
                                                                org_neighb = orig_points[
neighbours[ i ] ];

                                                                if ( placeNeighbourInItsArea(
org_coord, coord, org_neighb, neighb, rigidity[ coord_idx ])==true )
                                                                {
                                                                        new_deformed_vector[ j
][ neighbours[ i ] ]= neighb;
                                                                        moved_elements.push(
neighbours[ i ] );


     candidate_for_move.push( pair< unsigned long, vector< unsigned long > >( neighbours[
i ], neighb_total[ neighbours[ i ] ] ) );
                                                                }
                                                                // update its processing state
                                                                processing_state_vector[ j ][
neighbours[ i ] ] = true;

                                                        }

                                                }

                                        candidate_for_move.pop();
                                }

                                }
```

```
                        for ( unsigned int j(1); j!=index_point_to_move.size(); j++ )
                        {
                                new_deformed_vector[ j ] = new_deformed_vector[ 0 ];

                                new_deformed_vector[ j ][ index_point_to_move[ j ] ] =
penetration[ j ] ;
                                processing_state_vector[ j ][ index_point_to_move[ j ] ] =
true;
                                moved_elements.push( index_point_to_move[ j ] );
                                neighbours = neighb_total[ index_point_to_move[ j ] ];

                                // add moved element neighbours to the candidates for moving
list
                                candidate_for_move.push( pair<unsigned long, vector<
unsigned long > >( index_point_to_move[ j ], neighbours ) );


                                while( candidate_for_move.size() )
                                {
                                        coord_idx = candidate_for_move.front().first;

                                                neighbours =
candidate_for_move.front().second;

                                                coord = new_deformed_vector[ j ][ coord_idx
];


                                                org_coord = orig_points[ coord_idx ];

                                                for( unsigned int i(0), I(neighbours.size());
i!=I; i++ )
                                                {
                                                        if ( processing_state_vector[ j ][
neighbours[ i ] ] == false )
                                                        {

                                                                neighb = new_deformed_vector[
j ][ neighbours[ i ] ];
                                                                org_neighb = orig_points[
neighbours[ i ] ];

                                                                if ( placeNeighbourInItsArea(
org_coord, coord, org_neighb, neighb,  rigidity[ coord_idx ])==true )
                                                                {
                                                                        new_deformed_vector[ j
][ neighbours[ i ] ]= neighb;
                                                                        moved_elements.push(
neighbours[ i ] );

     candidate_for_move.push( pair< unsigned long, vector< unsigned long > >( neighbours[
i ], neighb_total[ neighbours[ i ] ] ) );

                                                                }
                                                                // update its processing state
```

```
                                                                processing_state_vector[ j ][
neighbours[ i ] ] = true;

                                                        }

                                                }

                                        candidate_for_move.pop();

                                }


                        }
// ************************************************************


                        new_deformed_points = new_deformed_vector[ 0 ];

                        bool neighb_moved=false;
                        Vec2f ini(0,0);
                        vector< Vec2f > point_moved(new_deformed_points.size(), ini);

                        for ( unsigned int j(0); j!=new_deformed_points.size(); j++ )
                        {
                                Vec3f vector_sum(0, 0, 0);
                                                                for ( unsigned int
k(1); k!=index_point_to_move.size(); k++)
                                {
                                        if (index_point_to_move[ k ]==j)
                                        {
                                                point_moved[ j ] = Vec2f(1,k);

                                        }
                                }

                                if (point_moved[ j ].x == 0)
                                {


                                        for ( unsigned int i(1); i!=nr_devices; i++ )
                                        {

                                                vector_sum += new_deformed_vector[ i ][ j ];

                                        }

                                        new_deformed_points[ j ] = vector_sum/(nr_devices-1);

                                }

                                else{
                                        int index = (int)point_moved[ j ].y;
                                        new_deformed_points[ j ] = new_deformed_vector[ index
][ j ];
                                }

                        }
```

```
        }
    }
```

7.3 Main X3D File

```
<Group>

 <ImportLibrary library='"chain.dll"'/>

     <Viewpoint position="0 140 60"/>
     <TimeSensor DEF='TS' loop='true' cycleInterval='2.0'/>

        <KeySensor DEF='KS'/>



        <RotTransform DEF="LEFT RIB1" center="3.00548 157.36839 -5.88361" angles="0.19 0.10 1">
                <Inline DEF='L1' url='./left_rib_1.x3d' />
                        <IMPORT inlineDEF='L1' exportedDEF='coordNode' AS='coordNodeLeft1'/>
        </RotTransform>


        <RotTransform DEF="RIGHT RIB1" center="-2.31494 157.62589 -5.51748 "  angles="0.19 0.10 -1">
                <Inline DEF='R1' url='./right_rib_1.x3d' />
                        <IMPORT inlineDEF='R1' exportedDEF='coordNode' AS='coordNodeRight1'/>

        </RotTransform>


        <RotTransform DEF="LEFT RIB2" center="4.55879 155.35158 -6.08521 "  angles="0.18 0.12 1">
                <Inline DEF='L2' url='./left_rib_2.x3d' />
                        <IMPORT inlineDEF='L2' exportedDEF='coordNode' AS='coordNodeLeft2'/>
        </RotTransform>


        <RotTransform DEF="RIGHT RIB2" center=" -2.06248 156.14081 -5.57082"  angles="0.18 0.12 -1">
                <Inline DEF='R2' url='./right_rib_2.x3d' />
                        <IMPORT inlineDEF='R2' exportedDEF='coordNode' AS='coordNodeRight2'/>
        </RotTransform>


        <RotTransform DEF="LEFT RIB3" center="2.22638 154.32162 -6.84439 "   angles="0.16 0.15 1">
                <Inline DEF='L3' url='./left_rib_3.x3d' />
                        <IMPORT inlineDEF='L3' exportedDEF='coordNode' AS='coordNodeLeft3'/>


        </RotTransform>


        <RotTransform DEF="RIGHT RIB3"  center=" -2.08931 153.88301 -7.14213"  angles="0.16 0.15 -1">
                <Inline DEF='R3' url='./right_rib_3.x3d' />
                        <IMPORT inlineDEF='R3' exportedDEF='coordNode' AS='coordNodeRight3'/>
```

```
   </RotTransform>


   <RotTransform DEF="LEFT RIB4"  center="2.33945 151.67632 -8.03701 " angles="0.14 0.17 1">
           <Inline DEF='L4' url='./left_rib_4.x3d' />
                   <IMPORT inlineDEF='L4' exportedDEF='coordNode' AS='coordNodeLeft4'/>
   </RotTransform>


   <RotTransform DEF="RIGHT RIB4"  center="-1.67923 152.06801 -7.37564 "  angles="0.14 0.17 -1">
           <Inline DEF='R4' url='./right_rib_4.x3d' />
                   <IMPORT inlineDEF='R4' exportedDEF='coordNode' AS='coordNodeRight4'/>
   </RotTransform>


   <RotTransform DEF="LEFT RIB5"  center="1.79046 149.48325 -8.4659 " angles="0.12 0.19 1">
           <Inline DEF='L5' url='./left_rib_5.x3d' />
                   <IMPORT inlineDEF='L5' exportedDEF='coordNode' AS='coordNodeLeft5'/>
   </RotTransform>


<RotTransform DEF="RIGHT RIB5"  center="-1.96822 149.65244 -8.40711 "  angles="0.12 0.19 -1">
           <Inline DEF='R5' url='./right_rib_5.x3d' />
                   <IMPORT inlineDEF='R5' exportedDEF='coordNode' AS='coordNodeRight5'/>
   </RotTransform>


   <RotTransform DEF="LEFT RIB6"  center="2.00545 147.55717 -9.39184 "  angles="0.12 0.20 1">
           <Inline DEF='L6' url='./left_rib_6.x3d' />
                   <IMPORT inlineDEF='L6' exportedDEF='coordNode' AS='coordNodeLeft6'/>
   </RotTransform>


<RotTransform DEF="RIGHT RIB6" center="-1.91813 147.45949 -9.46312 "   angles="0.12 0.20 -1">
           <Inline DEF='R6' url='./right_rib_6.x3d' />
                   <IMPORT inlineDEF='R6' exportedDEF='coordNode' AS='coordNodeRight6'/>
   </RotTransform>


    <RotTransform DEF="LEFT RIB7"  center=" 1.98886 144.69627 -9.96577"  angles="0.10 0.21 1">
           <Inline DEF='L7' url='./left_rib_7.x3d' />
                   <IMPORT inlineDEF='L7' exportedDEF='coordNode' AS='coordNodeLeft7'/>
   </RotTransform>


    <RotTransform DEF="RIGHT RIB7"  center="-1.87599 144.79588 -9.7722 "   angles="0.10 0.21 -1">
           <Inline DEF='R7' url='./right_rib_7.x3d' />
                   <IMPORT inlineDEF='R7' exportedDEF='coordNode' AS='coordNodeRight7'/>

           </RotTransform>

   <RotTransform DEF="LEFT RIB8" center=" 2.03204 142.04466 -10.10297"   angles="0.08 0.22 1">
           <Inline  DEF='L8' url='./left_rib_8.x3d' />
           <IMPORT inlineDEF='L8' exportedDEF='coordNode' AS='coordNodeLeft8'/>
   </RotTransform>

   <RotTransform DEF="RIGHT RIB8"  center="-1.85564 141.81032 -10.09296 "  angles="0.08 0.22 -1">
           <Inline DEF='R8' url='./right_rib_8.x3d' />
           <IMPORT inlineDEF='R8' exportedDEF='coordNode' AS='coordNodeRight8'/>

   </RotTransform>


   <RotTransform DEF="LEFT RIB9"  center=" 1.96802 139.23927 -10.15402"   angles="0.04 0.23 1">
           <Inline DEF='L9' url='./left_rib_9.x3d' />
```

```
                        <IMPORT inlineDEF='L9' exportedDEF='coordNode' AS='coordNodeLeft9'/>
        </RotTransform>


           <RotTransform DEF="RIGHT RIB9" center="-1.92039 139.36055 -10.0284 "   angles="0.04 0.23 -1"
>
                <Inline DEF='R9' url='./right_rib_9.x3d' />
                        <IMPORT inlineDEF='R9' exportedDEF='coordNode' AS='coordNodeRight9'/>
        </RotTransform>


        <RotTransform DEF="LEFT RIB10" center="2.05133 136.89429 -10.06546 "   angles="0.03 0.24 1">
                <Inline DEF='L10' url='./left_rib_10.x3d' />
                        <IMPORT inlineDEF='L10' exportedDEF='coordNode' AS='coordNodeLeft10'/>

        </RotTransform>


         <RotTransform DEF="RIGHT RIB10" center="-1.74045 136.9026 -9.89387 "   angles="0.03 0.24 -1" >
                <Inline DEF='R10' url='./right_rib_10.x3d' />
                        <IMPORT inlineDEF='R10' exportedDEF='coordNode' AS='coordNodeRight10'/>
        </RotTransform>


        <RotTransform DEF="LEFT RIB11" center=" 1.94558 133.43573 -9.40291"  angles="0.02 0.24 1" >
                <Inline DEF='L11'  url='./left_rib_11.x3d' />
                        <IMPORT inlineDEF='L11' exportedDEF='coordNode' AS='coordNodeLeft11'/>
        </RotTransform>


      <RotTransform DEF="RIGHT RIB11"  center=" -1.68157 133.25415 -9.18412"  angles="0.02 0.24 -1">
                <Inline DEF='R11'  url='./right_rib_11.x3d' />
                        <IMPORT inlineDEF='R11' exportedDEF='coordNode' AS='coordNodeRight11'/>
        </RotTransform>




        <RotTransform DEF="STERNUM" center='0.0 150  -5.88361'  angles='0.15 0.0 0'>
                <Inline DEF='ST' url='./sternum.x3d' />
                        <IMPORT inlineDEF='ST' exportedDEF='coordNode' AS='coordNodeSt'/>
        </RotTransform>

        <RotTransform DEF="CARTILAGE" center='0.0 150  -5.88361'  angles='0.15 0.0 0'>
                <Inline DEF='CART' url='./ligs.x3d' />
                        <IMPORT inlineDEF='CART' exportedDEF='coordNode' AS='coordNodeCart'/>
        </RotTransform>



    <Transform DEF='DIAPH' >
                <Inline DEF='DIAPHRAGM' url='./diaphragm4.x3d' />
                        <IMPORT inlineDEF='DIAPHRAGM' exportedDEF='deform' AS='deformableDiaph'/>
                        <IMPORT inlineDEF='DIAPHRAGM' exportedDEF='coordNode' AS='coordNodeDiaph'/>
                        <IMPORT inlineDEF='DIAPHRAGM' exportedDEF='TimeNode' AS='TS'/>
        </Transform>




        <Transform DEF="LEFT_LUNG" >
                <Inline DEF='LL'  url='./lung_left.x3d' />
                        <IMPORT inlineDEF='LL' exportedDEF='deform' AS='deformableLL'/>
```

```
                    <IMPORT inlineDEF='LL' exportedDEF='coordNode' AS='coordNodeLL'/>
                    <IMPORT inlineDEF='LL' exportedDEF='timeLL' AS='timeLL'/>

        </Transform>

        <Transform DEF="RIGHT_LUNG" >
                <Inline DEF='RL' url='./lung_right.x3d' />
                        <IMPORT inlineDEF='RL' exportedDEF='deform' AS='deformableRL'/>
                        <IMPORT inlineDEF='RL' exportedDEF='coordNode' AS='coordNodeRL'/>
                        <IMPORT inlineDEF='RL' exportedDEF='timeRL' AS='timeRL'/>

        </Transform>




         <Transform DEF="SPINE" >
                <Inline url='./spine.x3d' />
        </Transform>

        <RotTransform DEF='TOUCH_L10_D' dist='0.55, 1.5, -0.5, 12 , -20 ' />
        <RotTransform DEF='TOUCH_R10_D' dist='0.58, 1.5, -0.5, 12, -20 ' />
        <RotTransform DEF='TOUCH_L11_D' dist='0.51, 1.5, -5.8, 12, -20  ' />
        <RotTransform DEF='TOUCH_R11_D' dist='0.54, 1.5, -5.8, 12, -20 ' />


        <RotTransform DEF='TOUCH_R2_RL' dist='0.145, 3, 0, 12, -14' />

        <RotTransform DEF='TOUCH_R5_RL' dist='0.13, 2, 0, 12, -14' />


        <RotTransform DEF='TOUCH_L2_LL' dist='0.148, 3, 0, 12, -14' />

        <RotTransform DEF='TOUCH_L5_LL' dist='0.22, 2, 0, 12, -14' />



        <RotTransform DEF='TOUCH_D_RL' dist='0.65, 2.5, -2, 2, -6 ' />
        <RotTransform DEF='TOUCH_D_LL' dist='0.19, 2, -2, 2, -20 ' />



<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB1' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB1' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB2' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB2' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB3' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB3' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB4' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB4' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB5' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB5' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB6' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB6' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB7' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB7' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB8' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB8' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB9' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB9' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB10' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB10' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='LEFT RIB11' toField='rota'/>
```

```
<ROUTE fromNode='TS' fromField='time' toNode='RIGHT RIB11' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='STERNUM' toField='rota'/>
<ROUTE fromNode='TS' fromField='time' toNode='CARTILAGE' toField='rota'/>


<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_L10_D' toField='rota'/>
<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_R10_D' toField='rota'/>
<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_L11_D' toField='rota'/>
<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_R11_D' toField='rota'/>



<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_D_RL' toField='rota'/>
<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_D_LL' toField='rota'/>




<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_R2_RL' toField='rota'/>

<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_R5_RL' toField='rota'/>

<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_L2_LL' toField='rota'/>

<ROUTE fromNode='TS' fromField='startTime' toNode='TOUCH_L5_LL' toField='rota'/>



<ROUTE fromNode='coordNodeLeft10' fromField='point' toNode='TOUCH_L10_D' toField='coord1'/>
<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_L10_D' toField='coord2'/>

<ROUTE fromNode='coordNodeRight10' fromField='point' toNode='TOUCH_R10_D' toField='coord1'/>
<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_R10_D' toField='coord2'/>

<ROUTE fromNode='coordNodeLeft11' fromField='point' toNode='TOUCH_L11_D' toField='coord1'/>
<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_L11_D' toField='coord2'/>


<ROUTE fromNode='coordNodeRight11' fromField='point' toNode='TOUCH_R11_D' toField='coord1'/>
<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_R11_D' toField='coord2'/>


<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_D_RL' toField='coord1'/>
<ROUTE fromNode='coordNodeRL' fromField='point' toNode='TOUCH_D_RL' toField='coord2'/>
<ROUTE fromNode='coordNodeDiaph' fromField='point' toNode='TOUCH_D_LL' toField='coord1'/>
<ROUTE fromNode='coordNodeLL' fromField='point' toNode='TOUCH_D_LL' toField='coord2'/>


<ROUTE fromNode='coordNodeRight2' fromField='point' toNode='TOUCH_R2_RL' toField='coord1'/>
<ROUTE fromNode='coordNodeRL' fromField='point' toNode='TOUCH_R2_RL' toField='coord2'/>

<ROUTE fromNode='coordNodeRight5' fromField='point' toNode='TOUCH_R5_RL' toField='coord1'/>
<ROUTE fromNode='coordNodeRL' fromField='point' toNode='TOUCH_R5_RL' toField='coord2'/>


<ROUTE fromNode='coordNodeLeft2' fromField='point' toNode='TOUCH_L2_LL' toField='coord1'/>
<ROUTE fromNode='coordNodeLL' fromField='point' toNode='TOUCH_L2_LL' toField='coord2'/>

<ROUTE fromNode='coordNodeLeft5' fromField='point' toNode='TOUCH_L5_LL' toField='coord1'/>
<ROUTE fromNode='coordNodeLL' fromField='point' toNode='TOUCH_L5_LL' toField='coord2'/>
```

```
<ROUTE fromNode='TOUCH_L10_D' fromField='touch' toNode='deformableDiaph'
toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_L10_D' fromField='rigid_touch' toNode='deformableDiaph'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_L10_D' fromField='touch' toNode='LEFT RIB10' toField='touch'/>

<ROUTE fromNode='TOUCH_R10_D' fromField='touch' toNode='deformableDiaph'
toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_R10_D' fromField='rigid_touch' toNode='deformableDiaph'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_R10_D' fromField='touch' toNode='RIGHT RIB10' toField='touch'/>

<ROUTE fromNode='TOUCH_L11_D' fromField='touch' toNode='deformableDiaph'
toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_L11_D' fromField='rigid_touch' toNode='deformableDiaph'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_L11_D' fromField='touch' toNode='LEFT RIB11' toField='touch'/>


<ROUTE fromNode='TOUCH_R11_D' fromField='touch' toNode='deformableDiaph'
toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_R11_D' fromField='rigid_touch' toNode='deformableDiaph'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_R11_D' fromField='touch' toNode='RIGHT RIB11' toField='touch'/>



<ROUTE fromNode='TOUCH_D_RL' fromField='touch' toNode='deformableRL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_D_RL' fromField='rigid_touch' toNode='deformableRL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_D_RL' fromField='touch' toNode='deformableDiaph' toField='coord2'/>


<ROUTE fromNode='TOUCH_D_LL' fromField='touch' toNode='deformableLL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_D_LL' fromField='rigid_touch' toNode='deformableLL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_D_LL' fromField='touch' toNode='deformableDiaph' toField='coord'/>



<ROUTE fromNode='TOUCH_R2_RL' fromField='touch' toNode='deformableRL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_R2_RL' fromField='rigid_touch' toNode='deformableRL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_R2_RL' fromField='touch' toNode='RIGHT RIB2' toField='touch'/>

<ROUTE fromNode='TOUCH_R5_RL' fromField='touch' toNode='deformableRL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_R5_RL' fromField='rigid_touch' toNode='deformableRL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_R5_RL' fromField='touch' toNode='RIGHT RIB5' toField='touch'/>


<ROUTE fromNode='TOUCH_L2_LL' fromField='touch' toNode='deformableLL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_L2_LL' fromField='rigid_touch' toNode='deformableLL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_L2_LL' fromField='touch' toNode='LEFT RIB2' toField='touch'/>

<ROUTE fromNode='TOUCH_L5_LL' fromField='touch' toNode='deformableLL' toField='touch_point_to_update'/>
<ROUTE fromNode='TOUCH_L5_LL' fromField='rigid_touch' toNode='deformableLL'
toField='touch_rigid_points'/>
<ROUTE fromNode='TOUCH_L5_LL' fromField='touch' toNode='LEFT RIB5' toField='touch'/>


<ROUTE fromNode='LEFT RIB10' fromField='rib_points' toNode='deformableDiaph'
toField='points_to_follow'/>
<ROUTE fromNode='RIGHT RIB10' fromField='rib_points' toNode='deformableDiaph'
```

```
toField='points_to_follow'/>
<ROUTE fromNode='LEFT RIB11' fromField='rib_points' toNode='deformableDiaph'
toField='points_to_follow'/>
<ROUTE fromNode='RIGHT RIB11' fromField='rib_points' toNode='deformableDiaph'
toField='points_to_follow'/>



<ROUTE fromNode='deformableDiaph' fromField='coord_to_lung2' toNode='deformableRL'
toField='points_to_follow'/>
<ROUTE fromNode='deformableDiaph' fromField='coord_to_lung' toNode='deformableLL'
toField='points_to_follow'/>



<ROUTE fromNode='RIGHT RIB2' fromField='rib_points' toNode='deformableRL' toField='points_to_follow'/>

<ROUTE fromNode='RIGHT RIB5' fromField='rib_points' toNode='deformableRL' toField='points_to_follow'/>

<ROUTE fromNode='LEFT RIB2' fromField='rib_points' toNode='deformableLL' toField='points_to_follow'/>

<ROUTE fromNode='LEFT RIB5' fromField='rib_points' toNode='deformableLL' toField='points_to_follow'/>

<ROUTE fromNode='KS' fromField='actionKeyPress' toNode='deformableDiaph' toField='keyboard'/>
<ROUTE fromNode='deformableDiaph' fromField='pause_time' toNode='TS' toField='pauseTime'/>
<ROUTE fromNode='deformableDiaph' fromField='resume_time' toNode='TS' toField='resumeTime'/>
<ROUTE fromNode='KS' fromField='actionKeyPress' toNode='deformableLL' toField='keyboard'/>
<ROUTE fromNode='deformableLL' fromField='pause_time' toNode='timeLL' toField='pauseTime'/>
<ROUTE fromNode='deformableLL' fromField='resume_time' toNode='timeLL' toField='resumeTime'/>
<ROUTE fromNode='KS' fromField='actionKeyPress' toNode='deformableRL' toField='keyboard'/>
<ROUTE fromNode='deformableRL' fromField='pause_time' toNode='timeRL' toField='pauseTime'/>
<ROUTE fromNode='deformableRL' fromField='resume_time' toNode='timeRL' toField='resumeTime'/>



</Group>
```