TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Thomas Johann Seebeck Department of Electronics

Irina Konovalova 122183IAEMM

Dynamic thorax modelling system for bioimpedance simulation based on 4D human phantom

Master thesis

Supervisor: Rauno Gordon

Tallinn 2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

_____ _____

(kuupaev) Irina Konovalova

Table of contents

List of figures

Abstract

The aim of the present research was to find the way to simulate the bioimpedance of the 4D human body phantom taken from 4D XCAT in COMSOL. 4D XCAT is an application designed for medical images simulation. COMSOL simulates the physical effects.

First, XCAT simulates heartbeat and respiratory cycles. Then, the data from XCAT that consists of the coordinates of the organs' surfaces is exported to RAW file format. Each file corresponds to a particular moment of time and includes all the surfaces for the specific phase of breathing and heartbeat simulation. During the research the way to convert the XCAT RAW data into a commonly used surface file format has been found. The created scripts transform the quantity of points into STL file format. Since the obtained files are too complicated for the COMSOL simulation, they have been simplified. The result is a collection of STL files of each phase.

The next script imports the simplified and fixed files into COMSOL and performs the simulation. In order to build the live simulation the script creates COMSOL projects for each of the previously generated phases. Then the simulated current measurements are automatically saved into an output file.

As a proof of concept, the impact of heartbeat and respiration have been restored from the model of human body impedance.

Annotatsioon

Käesoleva uuringu eesmärk oli leida viis kasutada COMSOL'i bioimpedantsi 4D XCAT'i inimese keha mudelis simuleerimiseks. 4D XCAT on rakendus mis on mõeldud meditsiinilise kuvamise simuleerimiseks. COMSOL simuleerib füüsikalisi protsesse.

Kõigepealt XCAT simuleerib südame- ja hingamistsükkli. Pärast seda andmed XCAT'ist mis koosnevad organite pindade koordinaatidest eksporditakse RAW vorminguga failidesse. Iga fail vastab konkreetsele ajahetkele ja sisaldab kõik pinnad iga hingamis- ja südametsükkli simuleerimise faasi jaoks. Uuringu jooksul oli leitud meetod XCAT andmete mingi üldkasutatava tasandite failivorminguga andmetesse konverteerimiseks. Loodud skriptid teevad punktide hulgast STL vorminguga faile. Kuna saadud failid on COMSOL'i simulatsiooni jaoks liiga keerulised, neid lihtsustatakse. Tulemus kujutab endast STL failide kollektsiooni iga faasi jaoks.

Järgmine skript impordib lihtsustatud ja parandatud failid COMSOL'i ja teostab simulatsiooni. Selleks et saada ajaga muutuva sumulatsiooni, skript loob COMSOL projekti iga varem genereeritud faasi jaoks. Siis simuleeritud mõõtmised automaatselt salvestatakse tulemuste failisse.

Idee kinnitamiseks inimese keha mudeli impedansist taastatakse kopsude ja südame töö mõju.

1 Introduction and background

The new researches and achievements appear in the field of medicine almost every day. The inventions help people to detect diseases much earlier and consequently treat them with more probable success. The pre-symptomatic treatment is more effective than treatment when symptoms occur.

One of the most significant limitations while verifying innovative ideas is the fact that experiments on a live human are not possible. Therefore, the computer models of human body are essential researcher's instruments.

During the recent years many phantoms of living human body have been produced. Some of them describe only the visual characteristics of human body parts, others simulate effects of real medicine devices. However, some properties of human tissues have not been simulated yet.

The major focus of this research was to fulfill one of the lacks of currently available models of human body by modelling its bioimpedance.

From a high-level point of view the idea is to apply one of the well-known tools for physical systems modelling (COMSOL) to the human body models produced by a specific medical researcher's instrument (4D XCAT). While the idea seems to be quite straight-forward, its practical implementation is quite challenging since those tools have never been adapted to each other.

1.1 Project motivation

There are a lot of human body phantoms - virtual and physical - developed for medical imaging simulation. However, rare phantoms are intended to simulate the bioimpedance [19]. Intention of this work was to develop a system that allows simulating variations of dynamic human body impedance.

1.2 Research objectives

1) **Export the usable data from 4D XCAT**

XCAT is not designed to simulate the bioimpedance so the aim is to get the body shapes data in one of the common file formats.

2) Import the data into COMSOL

COMSOL is designed to simulate the physical effects but does not include any human body models and it can not generate the surfaces of a living organism. The aim is to make the XCAT data suitable for COMSOL simulation.

3) Simulation of the dynamic bioimpedance of human thorax

The aim is to simulate the bioimpedance measurement on dynamic thorax model. The dynamics of the signal would come from changes in anatomy - breathing and heartbeats - that the dynamic body model is exhibiting. The expected result of this research should be a dynamic bioimpedance signal where heartbeat and breathing can be observed.

2 Literature review

2.1 Methods and techniques for detection the heart activity

1) Electrocardiography

Electrocardiography is the process of recording the electrical activity of the heart over a period of time using electrodes placed on a patient's body. These electrodes detect the tiny electrical changes on the skin that arise from the heart muscle depolarizing during each heartbeat [8].

2) Positron emission tomography

Positron emission tomography (PET) is a nuclear medicine, functional imaging technique that produces a three-dimensional image of functional processes in the body. The system detects pairs of gamma rays emitted indirectly by a positron-emitting radionuclide (tracer), which is introduced into the body on a biologically active molecule. Three-dimensional images of tracer concentration within the body are then constructed by computer analysis [9].

3) Phonocardiography

Phonocardiography is diagnostic technique that creates a graphic record, or phonocardiogram, of the sounds and murmurs produced by the contracting heart, including its valves and associated great vessels. The phonocardiogram is obtained either with a chest microphone or with a miniature sensor in the tip of a small tubular instrument that is introduced via the blood vessels into one of the heart chambers. The phonocardiogram usually supplements the information obtained by listening to body sounds with a stethoscope (auscultation) and is of special diagnostic value when performed simultaneously with measurement of the electrical properties of the heart (electrocardiography) and pulse rate [10].

4) Echocardiography

Echocardiogram, often referred to as a cardiac echo or simply an echo, is a sonogram of the heart. Echocardiography uses standard two-dimensional, three-dimensional, and Doppler ultrasound to create images of the heart [11].

5) Coronary catheterization

A coronary catheterization is a minimally invasive procedure to access the coronary circulation and blood filled chambers of the heart using a catheter. It is performed for both diagnostic and interventional (treatment) purposes [12].

2.2 Types of models used to simulate the heartbeat and respiration

1) **Physical phantoms**

Physical phantom is a real model of human body parts and structures. The physical model allows performing the most precise simulations, but it is prohibitively expensive and has limited patient characteristic variations.

An example of such a phantom is the dynamic breathing phantom (TBP) made by inc. Radiology Support Devices (RSD) [14].

2) **Digital phantoms**

Digital phantoms provide a virtual model of the patient's anatomy and physiology. This type of phantoms is cheaper and more adaptable.

There are three general classes of digital phantoms:

- **Voxelized**

The voxelized models are based on the raw data, from CT scans, MRI imaging, or direct imaging through photography. Then the components of the body are segmented, or identified and separated from the rest. In voxelized phantom the result is cast into a 3D format [13].

For example the NORMAN phantom based on magnetic resonance images was developed by a team led by Dr. Dimbylow [16]. The male version of this phantom was created in 1996. In 2005 the team also created a female phantom.

- **Mathematical**

The mathematical model uses simple geometrical equations to define the body organs and structures.

The model is more suitable for deformation and it's geometry can be conveniently transformed to fit particular physical organ shapes and volumes. This type of phantom is realized by Non-Uniform Rational B-Spline (NURBS) method or polygonal mesh method, which are usually collectively called BREP methods. Compared to the voxel phantoms, BREP phantoms are better suited for geometry deformation and adjustment.

- **Hybrid phantoms**

The hybrid phantoms can produce both BREP and voxel data.

For example the 4D XCAT cardiac-torso phantom used in the research.

2.3 Analysis of changes in bioimpedance

The bioimpedance is measured when the electricity is applied from an external source outside the living organism under study [19]. The two parameters are able to affect the changes in bioimpedance: the blood volume and the air volume in lungs.

1) **Changes of blood volume**

The tissues of the body vary in their electrical conductivity. The electrical impedance of any part of the body depends on the proportions of the different types of tissues within it. Blood is the most conductive body tissue, and the electrical impedance of any part of the body will vary according to the amount of blood within it [21][22][23]. Variations in thoracic impedance occur with the filling and emptying of the heart [24].

Atzler and Lehmann were the first who in 1932 related the transthoracic changes in electrical impedance to events in the cardiac cycle [21]. In 1959 Nyboer improved and modified this work [22], and in 1974 Kubicek described a commercially available impedance plethysmograph [23]. Impedance plethysmography is measuring the variation of impedance of body segments due to variation in blood volume [20].

2) **Changes of air volume**

The electrical impedance of air is smaller than any body tissue. Therefore the changes in bioimpedance caused by breathing can be measured easier than cardiac signal. In most cases the aim is measure the heartbeat and the respiratory signal is accepted as corrupting [25].

3 Materials and tools used

This research simulates the changes in conductivity of human body caused by heartbeat and respiration. The aim of the research is to restore the signal of heartbeat and respiration.

The initial data is produced by 4D XCAT. The data allows reconstructing of the human body at any respiratory phase. However, XCAT does not have the ability to simulate the bioimpedance and also does not have any body tissues' parameters needed to simulate the bioimpedance.

On the other hand, COMSOL can simulate any electrical parameters but it does not have human body model.

So we need to simulate the human respiratory cycle in 4D XCAT and to import the result into COMSOL, where we can simulate the bioimpedance.

To perform this operation the MATLAB tool is used. It allows writing the code that converts the data produced by XCAT into the data accepted by COMSOL. MATLAB can also create the projects for COMSOL and get the results of the simulation. So, there is no need to use COMSOL graphical user interface each time.

The data obtained from XCAT is too complicated for COMSOL simulation. Therefore an additional tool, MeshLab, is used to simplify the phantom graphical data before importing it in COMSOL.

3.1 4D XCAT

4D extended cardiac-torso (XCAT) phantom for multimodality imaging research was developed by W. P. Segars, G. Sturgeon, S. Mendonca and Jason Grimes and B. M. W. Tsui. The 4D XCAT (which includes thousands of anatomical structures) can produce realistic imaging data [15].

3.2 MATLAB

MATLAB is the high-level language and interactive environment for numerical computation, visualization, and programming [1].

3.3 MeshLab

MeshLab is an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes.

The system is aimed to help the processing of the typical not-so-small unstructured models arising in 3D scanning, providing a set of tools for editing, cleaning, healing, inspecting, rendering and converting this kind of meshes [2].

3.4 COMSOL

COMSOL Multiphysics is a finite element analysis, solver and Simulation software / FEA Software package for various physics and engineering applications, especially coupled phenomena, or multiphysics [3] [4].

The LiveLink interface is based on the COMSOL client/server architecture. A COMSOL thin client is running inside MATLAB and has access to the COMSOL API through the MATLAB Java interface. Model information is stored in a model object available on the COMSOL server. The thin client communicates with the COMSOL server, enabling you to generate, modify, and solve COMSOL model objects at the MATLAB prompt [18].

3.5 STLView

STLView is a free program for viewing of graphical files in STL format [17]. It can not transform the STL files, but it works faster than COMSOL or MeshLab and it has been used for visualizing of STL files and finding mistakes of the surfaces.

4 Implementation

The XCAT produces the 4D data of human body as an amount of so-called "frames". Each frame includes the complete body phantom in a specific moment of time. The number of frames, the time of simulation and the time period between frames are configurable.

COMSOL works only with static models. Therefore, in order to receive the information about the changes in bioimpedance that take place in living body it has been decided to create a certain number of frames by XCAT and to process them one by one in COMSOL.

4.1 Producing the detailed model of human body by 4D XCAT

The initial point of the research is to create the phantom of human body. The idea was to simulate the signal of moving heart and lungs, so the whole cycle of heartbeat and respiration was needed.

By default the length of one heartbeat in XCAT is 1 second, which is typical for a healthy human. The length of one respiratory cycle is 5 seconds that is also quite typical. Therefore, in order to cover both respiratory and heartbeat cycles the length of the simulation should be at least 5 seconds.

The following parameters were used:

- Simulation length: 6 seconds

- Number of frames: 100

In the other words, time gap between frames: 0.06 seconds

As long as 4D XCAT was designed to be a tool for medical imaging simulation, the output data is similar to a real medical device result. Medicine applications usually work with 3D matrixes data. So the output data of the XCAT are ".bin" files with 3D matrix inside. Each point of the matrix contains the value of attenuation characteristic got by hypothetic medicine device.

However, COMSOL does not accept the 3D matrixes as an input file. It operates only with surfaces, so the way to convert the data into surface has to be found.

For better understanding of the 3D matrixes' contents the MATLAB script "bodybuilder.m" was implemented. The script prints out a slice of the matrix where symbol "@" means the given material. All possible values of the material used in future scripts have also been found with the modified version of this code.

1) The first attempt

The first opportunity to build the surface data for COMSOL was to design the code that converts the 3D matrix into surfaces. The MATLAB script "readbin.m" was written for this purpose. It reads the matrix's values one by one, ignores the voxels with the same value as previous but saves the coordinates of the points where values are changing to the value it was given. In that way the algorithm detects coordinates of a surface that wraps a body of a given material. Than it calls the script "surf2stl.m". This script is a free MATLAB script written by Bill McDonald [26]. The script constructs the STL file basing on its points' coordinates. It also used to add the "bottom" points of the 3D matrix to the final file:

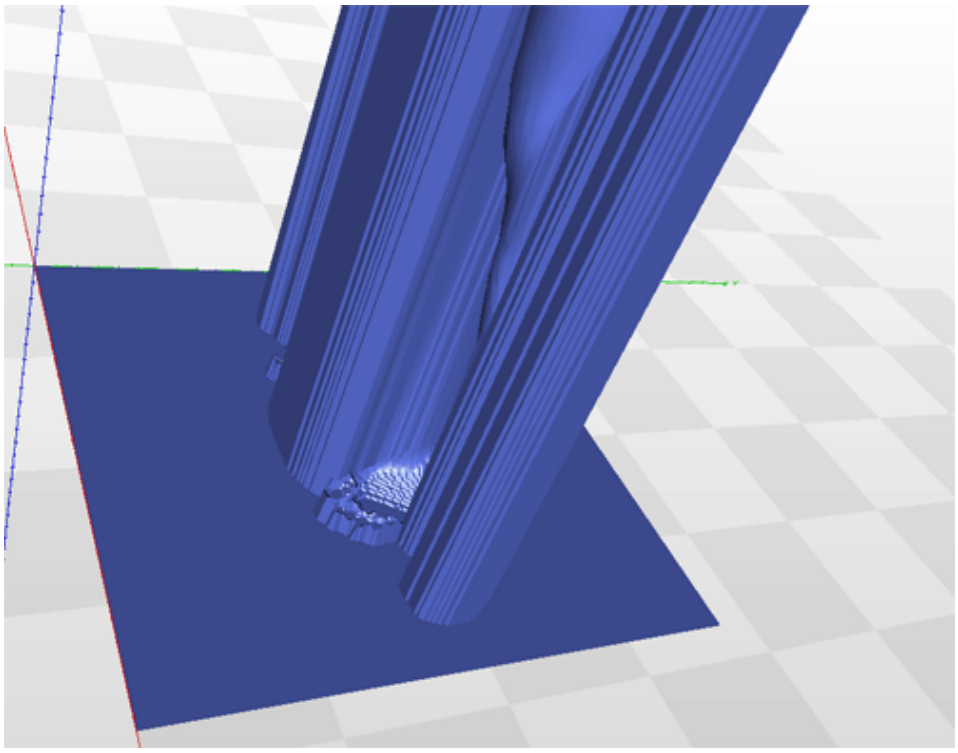

Figure 1 The surface with "bottom" points

The script has been changed for excluding those points and to build only the organ's surface. The problem is that the script "readbin.m" can only find the front surface of an organ. The resulting STL file includes only a part of an organ:

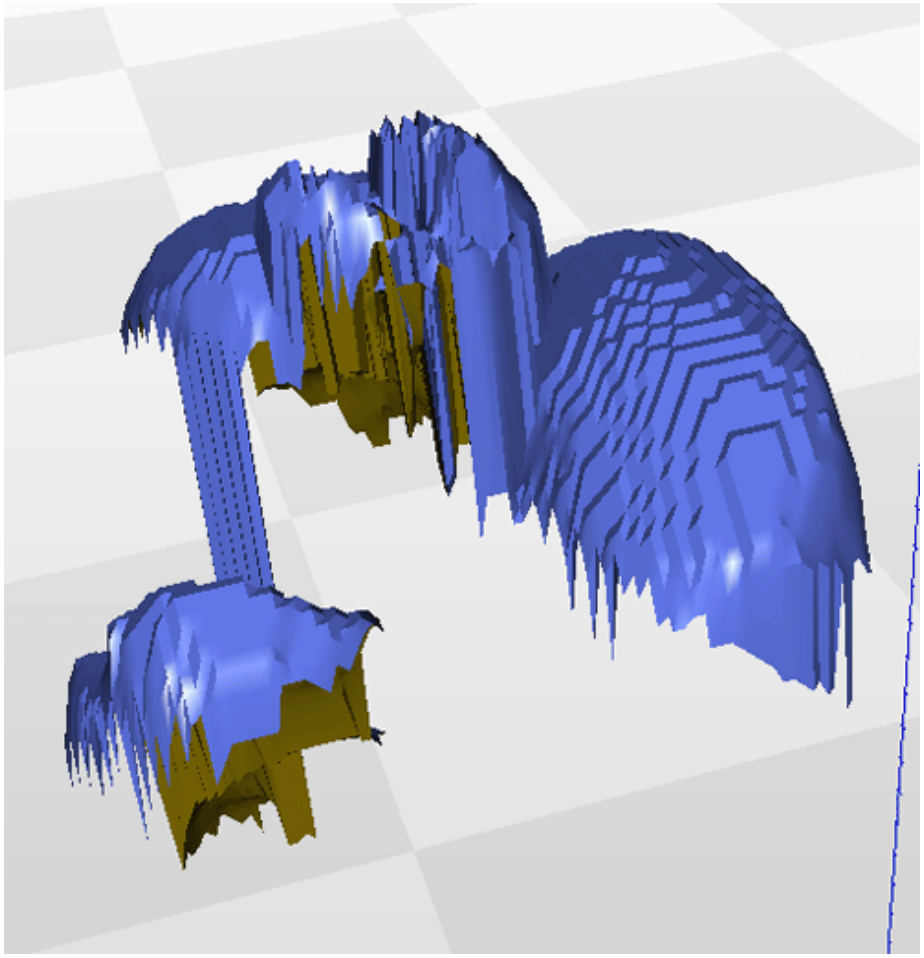

Figure 2. The "front" surface of an organ

To solve that problem the MATLAB function "isosurface" was used. It reconstructs the surface from given 3D matrix. The MATLAB script "isosurf.m" has been implemented. To find the points of the surface the script uses the function "isosurface" and calls the free script "stlwrite.m" to write the STL file. The "stlwrite" was written by Sven Holcombe [27]. The script reconstructed the surface successfully.

However, XCAT produces the files where each point of the human body is connected to the attenuation characteristics instead of the body part. So the organs with the same attenuation characteristics have the same value in the file and it is not possible to distinguish the organs. The solution was treated as inappropriate. A sample file produced by "isosurf":

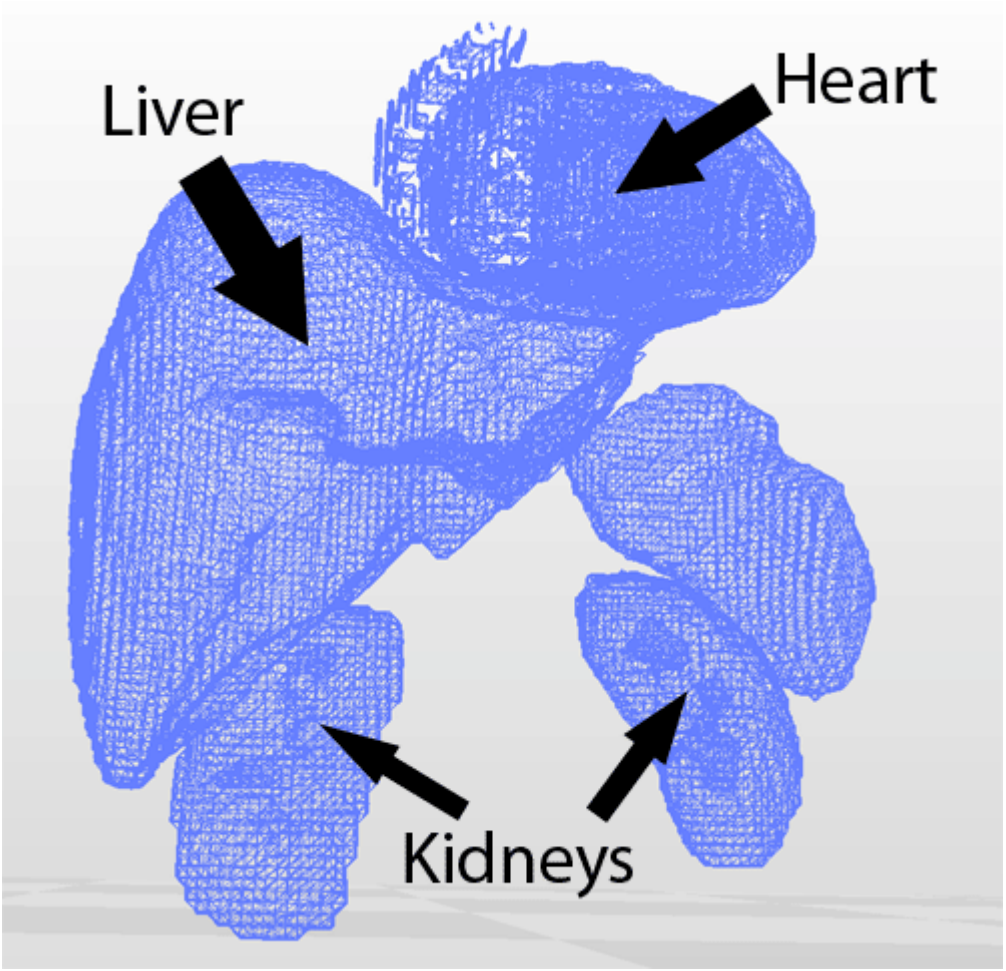

Figure 3. Organs with the similar attenuation characteristics

2) The working solution

Finally, another opportunity to get the surface data for COMSOL was found. XCAT has an undocumented parameter that allows saving the surface data in addition to ".bin" files. To activate this mode the following parameter needs to be added to the XCAT parameter file: "mesh_save = 1". As the result of using this parameter XCAT also saves a RAW file. The RAW file consists of a set of the organs' surface triangles. These triangles can be converted into the surface file accepted by COMSOL.

4.2 Creating the surface file acceptable for COMSOL import

The acceptable import file formats for COMSOL are: 3D CAD file, STL/VRML file, GDS, NETEX-G and ODB++. It was decided to use the STL file format, because it is widely used, manageable and does not include unnecessary details like color, texture or other common CAD model attributes [5].

An STL file describes a raw unstructured triangulated surface by the unit normal and vertices of the triangles using a three-dimensional Cartesian coordinate system.

To create an STL file from the RAW file generated by XCAT a MATLAB script "build_data_stl.m" has been designed. It also uses the script files "make_stl.m" and "config.m".

"config.m" is the configuration file and includes the parameters for all code files. In particular, the "build_data_stl.m" takes from the configuration file the number of frames and the list of body parts to be converted.

To define the parameters for creating the STL from RAW the following "config.m" parameters must be specified:

The number of start frame (for example: 1),

```
cfg_frame_start = 1;
```

The number of end frame (for example 100):

```
cfg_frame_end = 100;
```

The required body parts (for example: body, heart and organs):

```
cfg_body_part = {'body', 'heart', 'organs'};
```

"build_data_stl.m" calls the function "make_stl.m" for each of the selected frames and body parts. Since the whole body is not needed for the research, "make_stl.m" cuts off all unnecessary parts of the surfaces. For that purpose it uses the "config.m" parameters `cfg_zmin` and `cfg_zmax`. The first one chooses where to cut the body from the top, the second where to cut it from the bottom (following Z axes). The chosen parameters were:

```
cfg_zmin = 12;
cfg_zmax = 17;
```

As a result a number of STL files is generated, that describe all surfaces for all chosen organs, frames and body area. Here is an example of the lungs and thorax STL files opened in STLView:

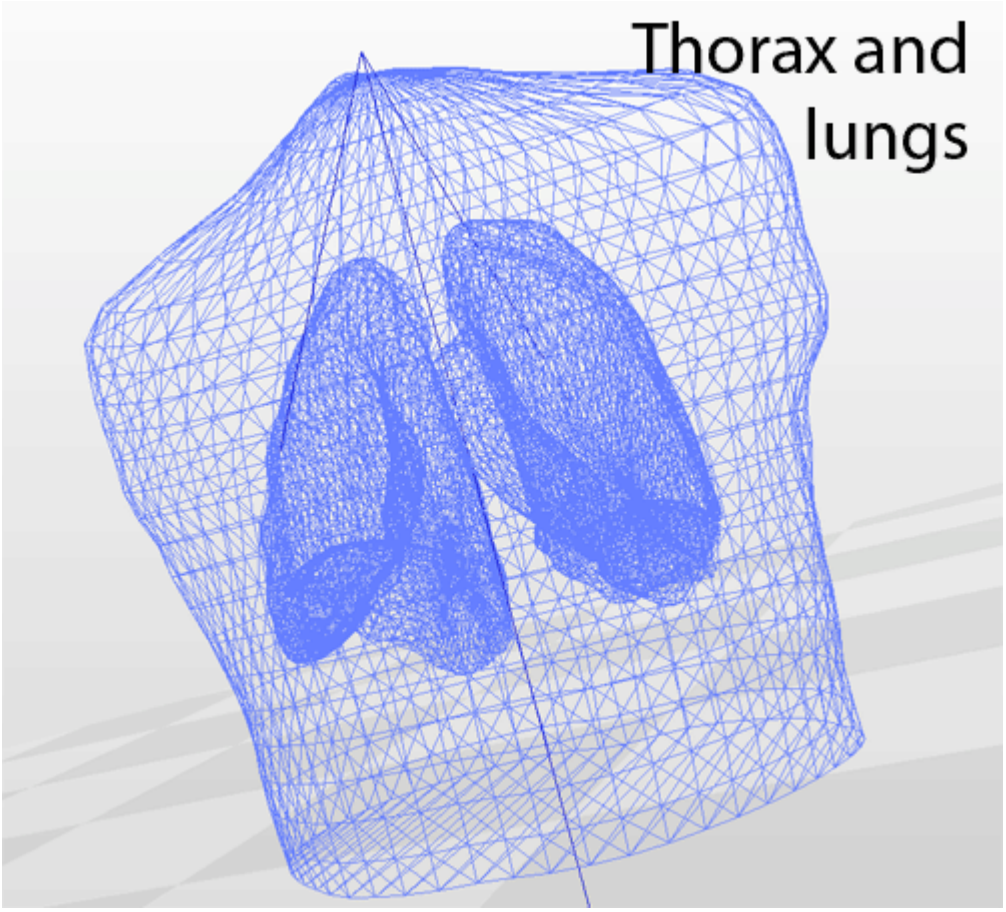

Figure 4. Thorax and lungs surfaces

Those files can be opened in COMSOL. However, the created surfaces have inaccuracies caused by initial RAW data mistakes and rounding. Because of the inaccuracies, holes, non-manifoldness and other problems may appear. On another hand, COMSOL for its simulation strictly requires water-proof surfaces. So, although COMSOL can import the created files as is, it is not possible to use them for a simulation.

In addition, the resulting files are too detailed to be processed by COMSOL. When COMSOL attempts to build the geometry using too sophisticated surfaces it stops working (hangs or even crashes).

Therefore, the surfaces described in STL files should be fixed and simplified.

4.3 Fixing and simplifying the STL files

To fix and to simplify the STL files the open source program MeshLab was chosen. It has a command-line interface so it can be used without the graphical user interface that makes it easier to automate the fixing and simplification via scripting [6].

In MeshLab every used filter can be saved in form of ".mlx" file. Later it can be called without starting the whole MeshLab user interface. To perform the conversion of all files the needed filters were used manually. The next step was a command line script "run.cmd" that automatically takes all files with names matched to a specific pattern and applies the chosen filters to all of them. It is also possible to define specific filters file for any particular organ. If there is a filters file ".mlx" which name coincides with any name of being converted STL file, the "run.cmd" will use this filters file to convert the files with the coincident names. For example the filters' file "body_chest_surface.mlx" was used to convert the files called "xxx_body_chest_surface.stl" where "xxx" is the frame number. If the filters file with selected organs' name does not exist, the default filters file "default.mlx" is used.

The names and order of MeshLab filters used:

1. Remove Duplicated Vertex
2. Remove Duplicate Faces
3. Quadric Edge Collapse Decimation
4. Select Self Intersecting Faces
5. Delete Selected Faces
6. Select non Manifold Edges
7. Delete Selected Faces
8. Select non Manifold Vertices
9. Delete Selected Faces
10. Close Holes

STL thorax file before the simplification:

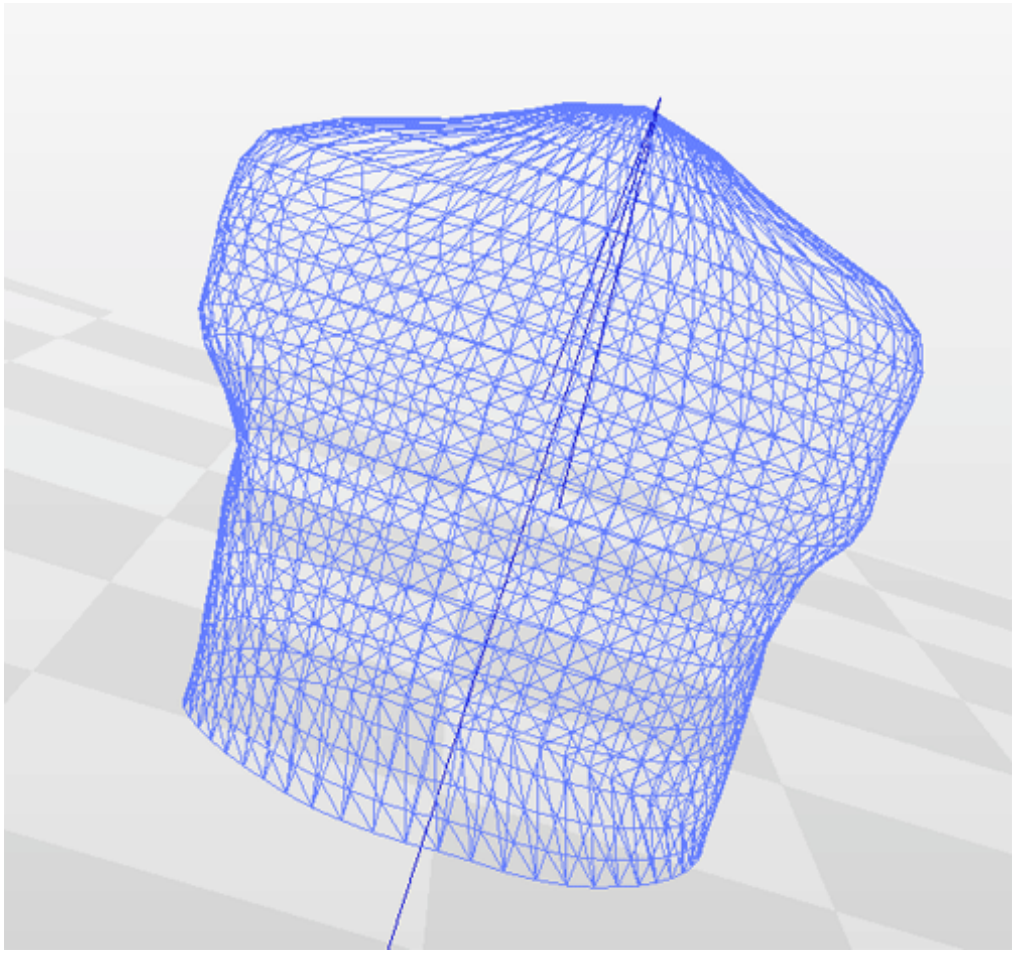

Figure 5. Thorax file before simplification

STL thorax file after the simplification:

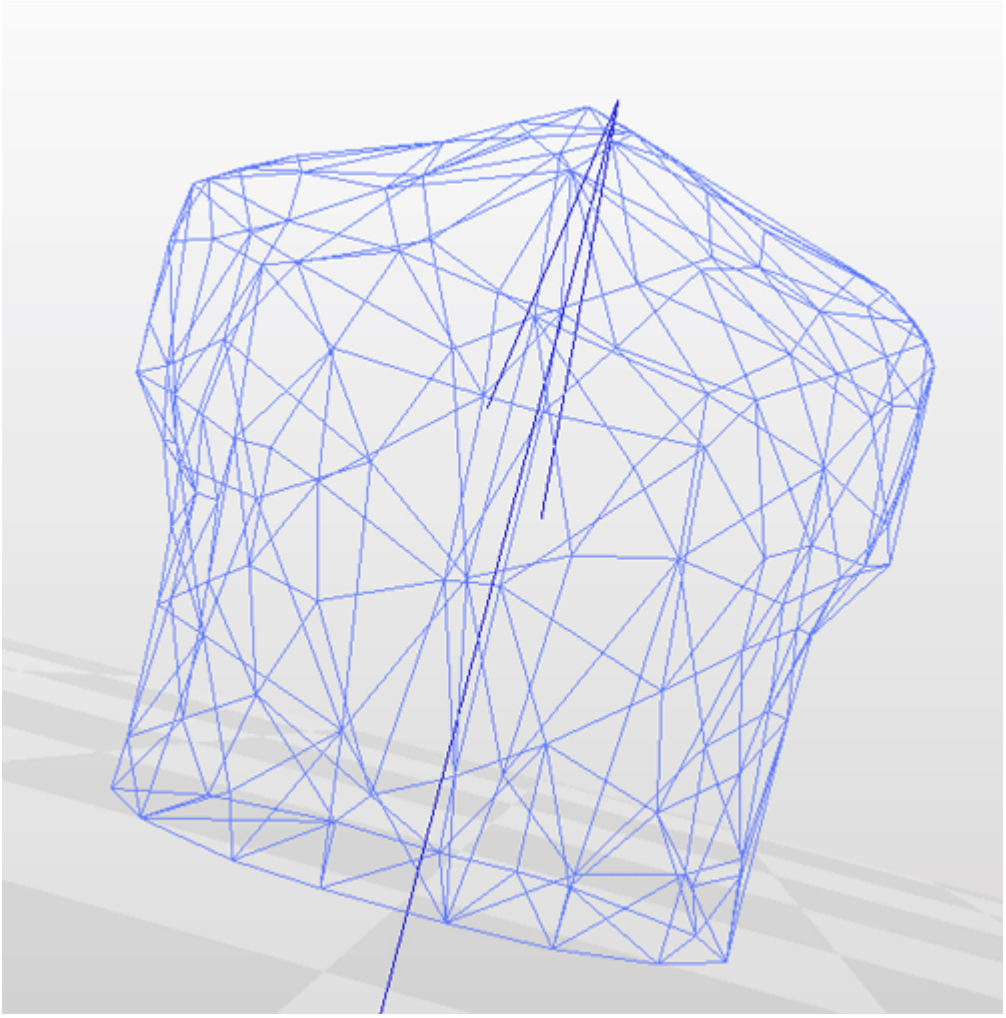

Figure 6. Thorax file after simplification

The result of MeshLab work is saved as the STL file with former name and the "fixed" postfix. These files are in most cases acceptable by COMSOL and can be used in simulation. Some surfaces have major defects which can not be automatically repaired by MeshLab filters. However, the amount of acceptable files is big enough to conduct the research.

4.4 Using MATLAB to control simulations in COMSOL

Creating 100 COMSOL projects manually would take incredible time, so the integrated "COMSOL Multiphysics with MATLAB Scripting" was used. This tool allows MATLAB to use the COMSOL libraries. As the result, the COMSOL projects can be created, changed and simulated automatically without starting the COMSOL graphic user interface. The simulation of COMSOL projects with MATLAB can also be easily automated and takes much less computer resources.

The MATLAB script that creates and simulates the COMSOL project file is called "build_data_comsol.m". The script "config.m" is also used for configuration.

a. Geometry

The first task was to create the geometry. The final geometry includes the human (thorax and inner organs like heart and lungs) and electrodes. The human body is imported as a set of STL files.

Electrodes appeared to be an additional difficulty. In theory, they are simple figures that could be defined by COMSOL primitives. There were several attempts to create the electrodes in this research.

1) The first attempt

The first attempt was to avoid using any additional figures for electrodes. Instead of applying the potential to the electrode it can be applied directly to the thorax boundaries.

It is possible to apply the electric potential to any manually selected boundary. Each boundary has some index. When user selects the boundary manually, COMSOL associates the electric potential with the index of chosen boundary. However, the indexes for boundaries are generated randomly and this process can not be controlled by MATLAB script. So the ball selection was used to select the boundary:

Geometry -> Selections -> Ball selection

The ball selection selects every boundary it touches. Later the electric potential was applied to the selected boundaries. The system worked, but as the configuration of the surfaces was different in every frame, the area of the selection is changed either. So it was not possible to compare the values received from different frames. The solution considered to be inappropriate.

2) The second attempt

The second try was to use the COMSOL created cylinders for electrodes:

Geometry -> Cylinder

However, another problem appeared.

Each object in COMSOL is usually isolated from the others. As the aim was to simulate the conductivity of the model, the parts of it should not be isolated from each other. To avoid the isolation the option "Form union" should be used:

Geometry -> Booleans and partitions -> Union

The problem was that in case the geometry includes the intersecting imported and created surfaces, the Union can not be built. The error message appears:

"Unsupported operation on this type of object."

So the attempt with created cylinders did not work.

3) The third attempt

The third try was to use instead of cylinders the COMSOL created spheres:

Geometry -> Sphere

It was possible to form the Union with created spheres. Although the same error as in second attempt in case of some frames appeared, the majority of the frames were processed. The electric potential and ground were later applied to halves of the spheres with ball selection. The simulation worked, but as in the first attempt the configuration of the surfaces changed in every frame. The sphere sank every frame into the thorax to a different level and the area of applied electric potential was different every time. So it was not possible to catch the difference between values caused by respiration and heartbeat.

That solution also considered to be inappropriate.

3) The working approach

COMSOL fails only if geometry includes both the created and imported surfaces. To solve the problem the files with electrodes were created separately by COMSOL and exported into two separate STL files "cyl_right.stl" and "cyl_left.stl". Later they were imported into each project as well as other STL files.

The cylinders were partly set into the thorax. The inner part of the cylinder had the same material properties as the thorax and the outer part had the material of an electrode.

To select the boundaries for potential applying the ball selection was used with parameter "Output Entities: Entity inside ball" and was put in a position when only the base of the cylinder was inside the ball selection.

The ball selection:

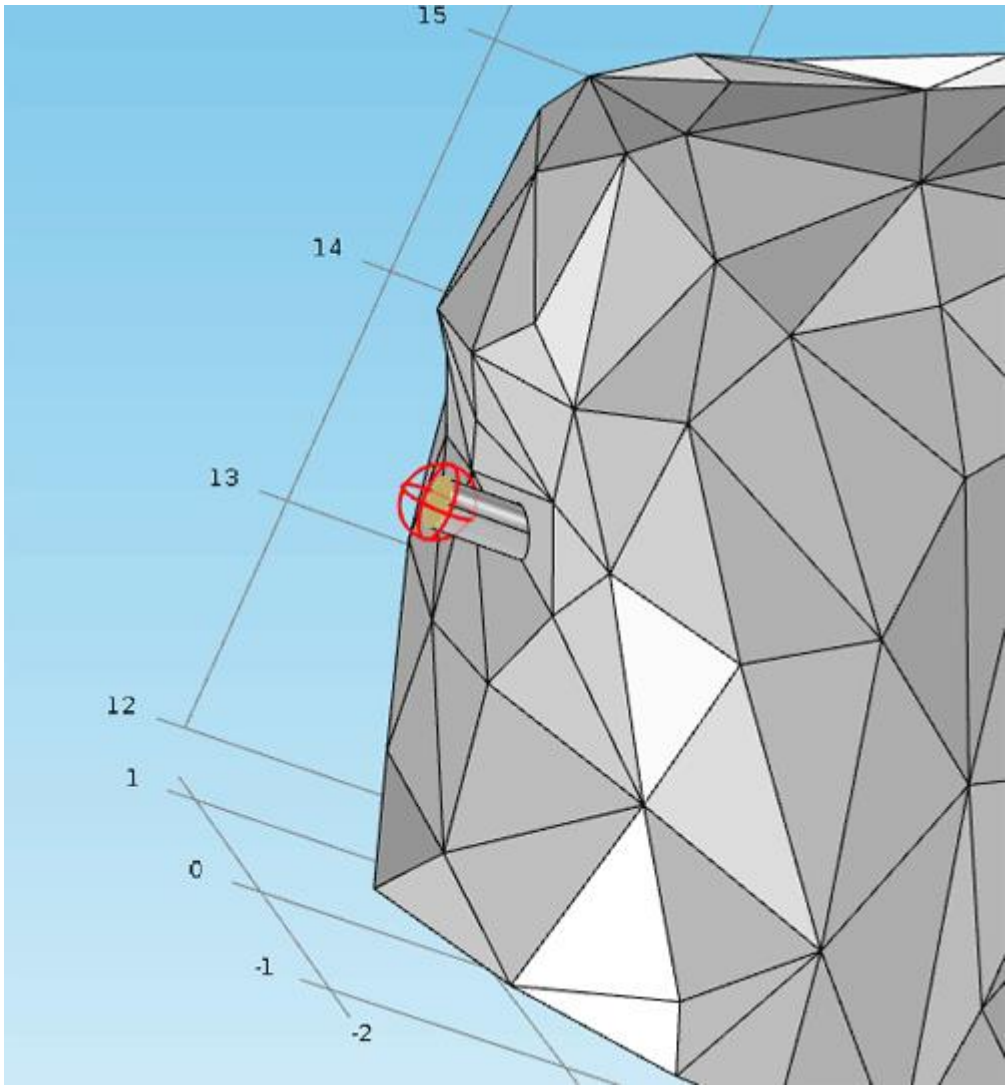

Figure 7. The "ball selection"

Then the geometry could be built.

b. Materials

Each object in COMSOL should be associated with a material that describes the needed simulation properties of the object. For the research the electrical conductivity and the relative permittivity values of human body tissues were taken from the webpage "Dielectric Properties of Body Tissues" [7]. The electrodes were meant to be made from copper, but an unexpected issue occurred.

It was suspected that COMSOL simulates the currents incorrectly when the difference in conductivity between used materials is too big.

Here are two pictures of the same geometry with the same parameters. The only difference was the electrical conductivity of the bottom cubes. The conductivity value of the upper cube is in both cases the same (0.7[S/m]).

The simulation with electrical conductivity = 5.998e7[S/m]:

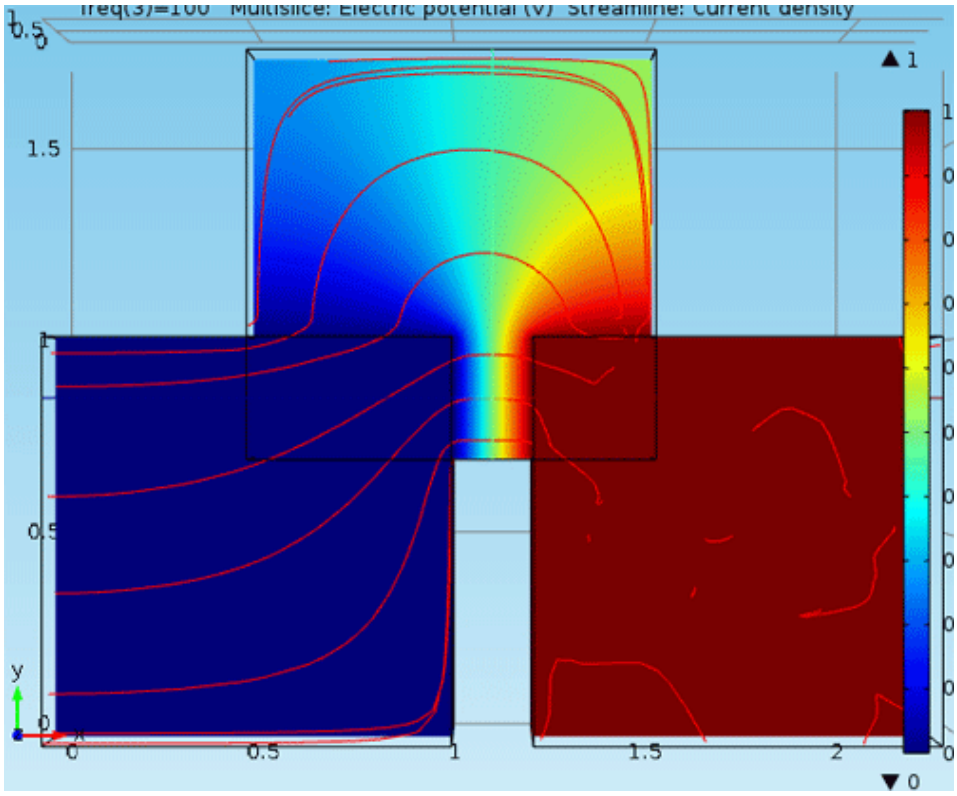

Figure 8. The simulation plot with bigger electrical conductivity difference between materials

The same simulation with electrical conductivity = 5.998e1[S/m]:

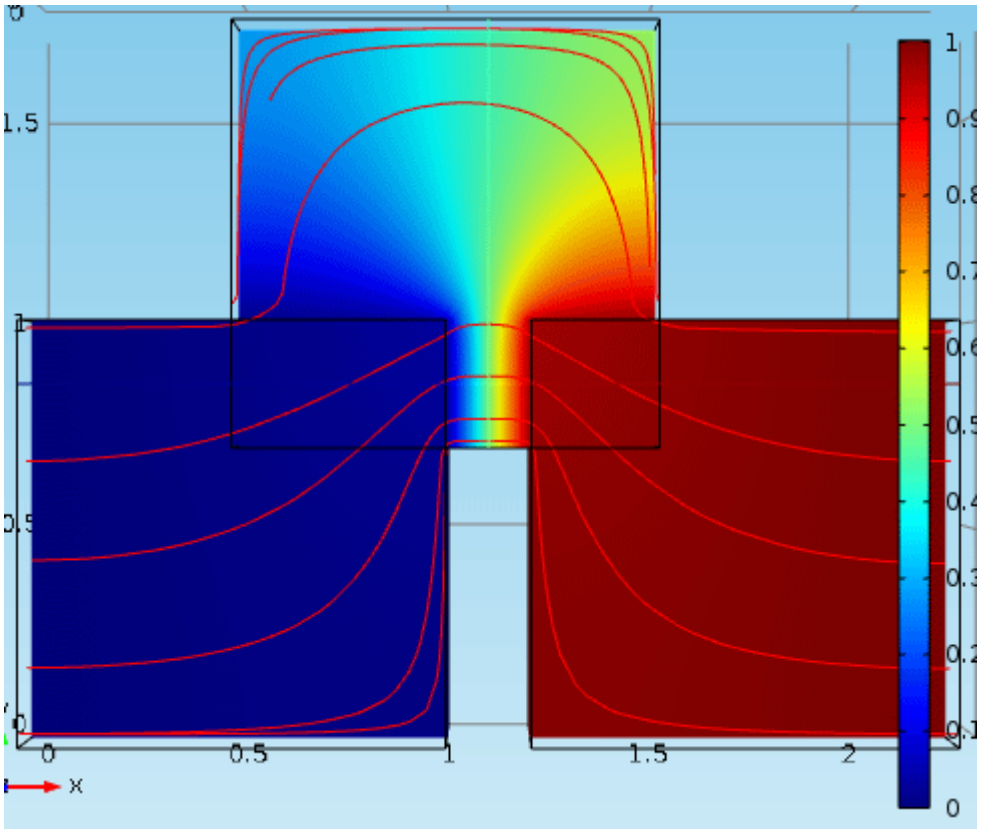

Figure 9. The simulation plot with smaller electrical conductivity difference between materials

In first case the streamlines are broken. It seems to be a COMSOL issue. To avoid the suspicious results the conductivity was decreased.

The taken parameters of electrodes (corresponds to copper):

Electrical conductivity: 5.998e1 S/m

Relative permittivity: 1

The material can be manually associated with boundary, edge, point or domain. During the STL import, each organ defines a domain and is being associated with a corresponding selection. Later, the selection is assigned by a material.

The selection of domains and materials can fail if the STL file normals' direction is inverted. Usually normals' direction is inside-out, but if it is not true for a particular organ file, all normals are inverted on the phase of simplification by applying an additional "Invert Faces Orientation" MeshLab filter.

c. Electric Currents

Electric potential and ground should be connected to the ball selections created in Geometry.

The value of Electric potential was 0.05 V. The modeled thorax has no skin, so the value of electric potential is smaller than allowable one for a real human.

d. Mesh

The mesh was used with the following parameters:

Sequence type: Physics-controlled mesh

Element size: Coarse

The mesh phase is the most fault-probable. Very small changes in geometry can cause a fail of meshing.

e. Study

In study settings the frequency 10 Hz has been selected. In Results -> Derived values two Surface Integration are created. Both values measure the summary current flows through a surface. The first one measures the input (Electric Potential) boundary current, the second one measures the output (Ground) boundary. The numbers of the first and the second integration should match (a small difference may be caused by calculation inaccuracy) .

In Results -> Plots the Current density is chosen. The expression is

sqrt(ec.Jx^2+ec.Jy^2+ec.Jx^2)

Also in Plots the Streamline should be selected.

The resulting plot:

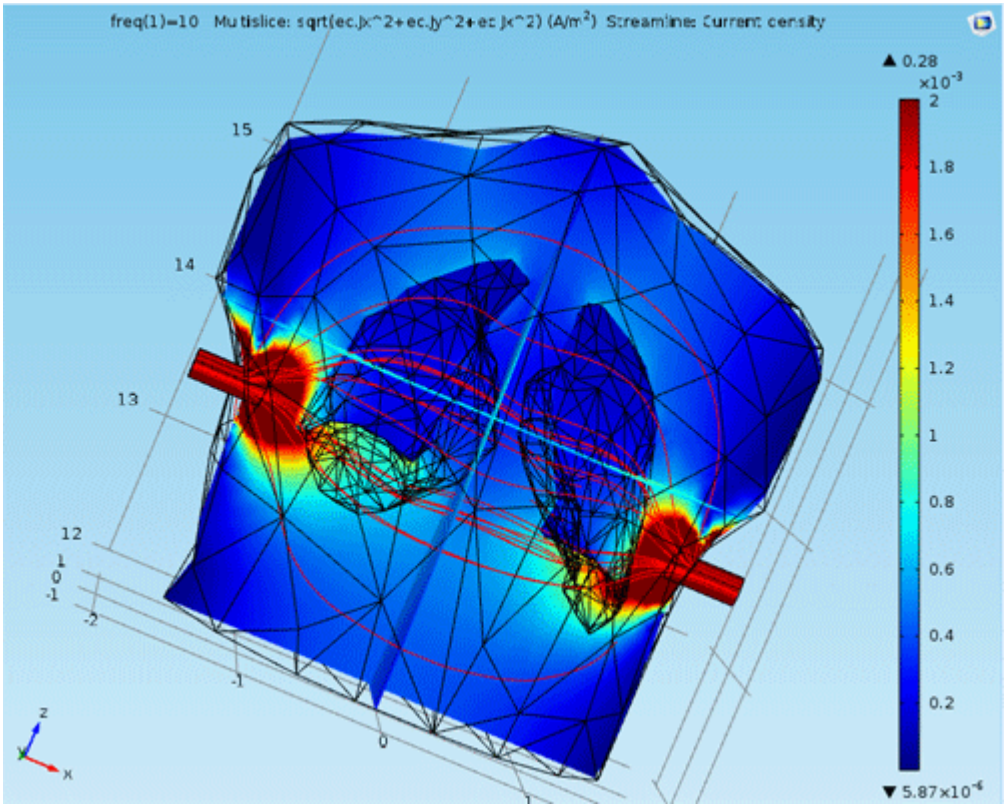

Figure 10. The resulting plot of the simulation

f. Processing the large quantity of files

Even when all simplifications and fixes are made, some projects can not be built or simulated by COMSOL. However, some missing points do not violate the reconstruction of the signal.

Only when too many frames have failed, the reconstruction is impossible. The more complicated projects we try to build, the more frames may fail. The project can be complicated because of too many included surfaces (organs), too uneven surfaces or intersection of the surfaces at too small angle to each other.

Some surfaces can not be fixed by automatic simplification and fixing. The projects with these surfaces also fail.

So to generate enough data to restore the respiration or heat beating signal the generated projects should be simple enough.

4.5 The final tutorial

Here is the final action sequence needed to import the data from 4D XCAT into COMSOL:

1) Perform the XCAT simulation with the line "mesh_save = 1" added to the parameter file.

2) Convert the received RAW file into SLT files by script "build_data_stl.m" (the scripts "make_stl.m" and "config.m" should be included)

3) Simplify surfaces by using "run.cmd" batch file. Inspect the run..cmd file for specifying the processed organs and filters.

4) Perform the simulation in COMSOL by MATLAB script "build_data_comsol.m" (the script "config.m" should be included)

5. Results and analysis

5.1 Receiving the data from 4D XCAT

The first objective to receive the usable data from 4D XCAT has been attained. The RAW files with collection of surface points fit the requirements. The data can be converted into common STL file format.

5.2 Importing the data to COMSOL

The data converted to STL file was imported into COMSOL. So the second objective has also been gained.

5.3 Simulation of the dynamic bioimpedance

The more complicated projects are generated, the less values of the signal it is possible to receive. So the first project includes only thorax, lungs and electrodes. The parameter for mesh was "coarser".

The surfaces of electrodes, lungs and thorax opened in COMSOL:

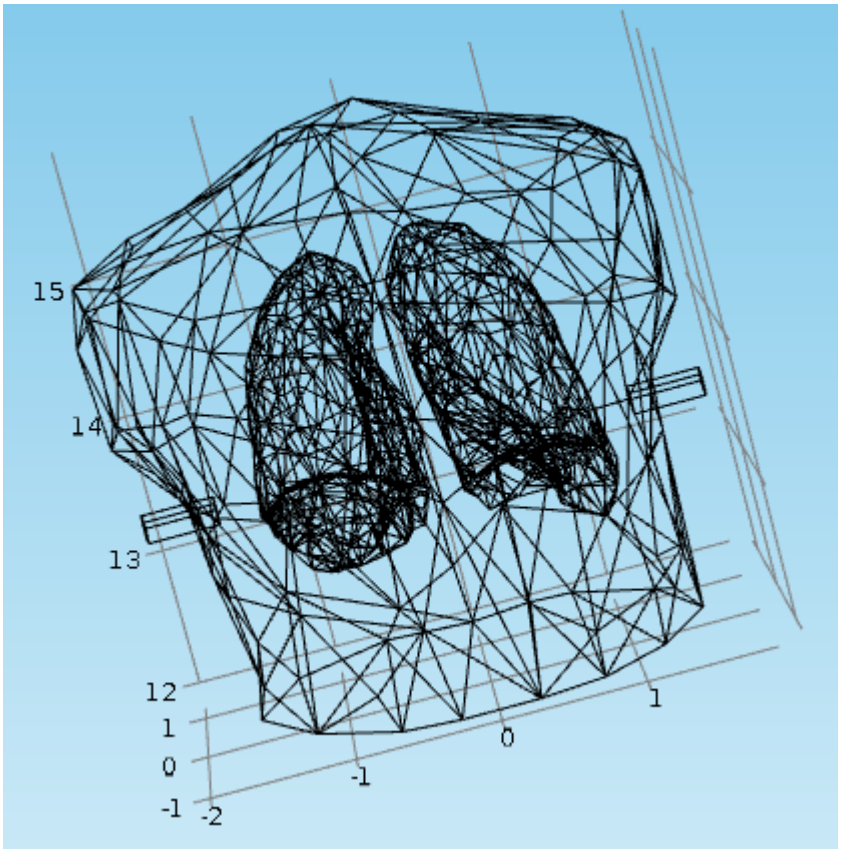

Figure 11. The surfaces of electrodes, lungs and thorax

The resulting simulated measures:

FRAME	L. CURRENT
1	0.002138
2	0.002148
3	-
4	0.002123
5	0.002143
6	-
7	-
8	0.002122
9	0.002149
10	0.002107
11	0.002099
12	0.002114
13	-
14	-
15	0.002081
16	0.002100
17	-
18	0.002079
19	0.002096
20	0.002117
21	0.002072
22	-
23	0.002076
24	-
25	0.002113
26	0.002062
27	-
28	0.002073
29	0.002098
30	0.002080
31	0.002089
32	-
33	-
34	0.002080
35	-
36	0.002078
37	0.002094
38	-
39	0.000080
40	0.002080
41	0.002067
42	0.002079
43	0.002088
44	0.000078
45	0.002072
46	0.002064
47	0.002059
48	0.002073
49	0.002065
50	0.000079
51	0.002073
52	0.002067
53	0.002097
54	-
55	0.002092

56	0.002093
57	0.002069
58	0.002091
59	0.002111
60	-
61	0.002091
62	0.002098
63	0.002103
64	0.002095
65	0.002112
66	0.002159
67	0.002129
68	0.002122
69	-
70	-
71	0.002147
72	0.002117
73	0.002140
74	0.002135
75	-
76	-
77	0.000078
79	0.002106
80	0.002150
81	0.002164
82	-
83	0.002151
84	0.002134
85	0.002149
86	0.002161
87	0.002155
88	0.002129
89	0.002135
90	0.000079
91	0.002092
92	0.002163
93	0.002116
94	0.002120
95	0.002139
96	-
97	0.002169
98	0.000080
99	0.002100
100	0.002066

The values from projects where geometry or mesh built has failed are missing because the script was not able to save the data (marked with red). The projects where electric potential settings failed or some other problems occurred are also inauthentic (marked with yellow). Those values have been excluded from the consideration.

After removing all the failed values 72 frames of initial 100 remain. It is quite enough to restore the signal.

There is the graph for these values:

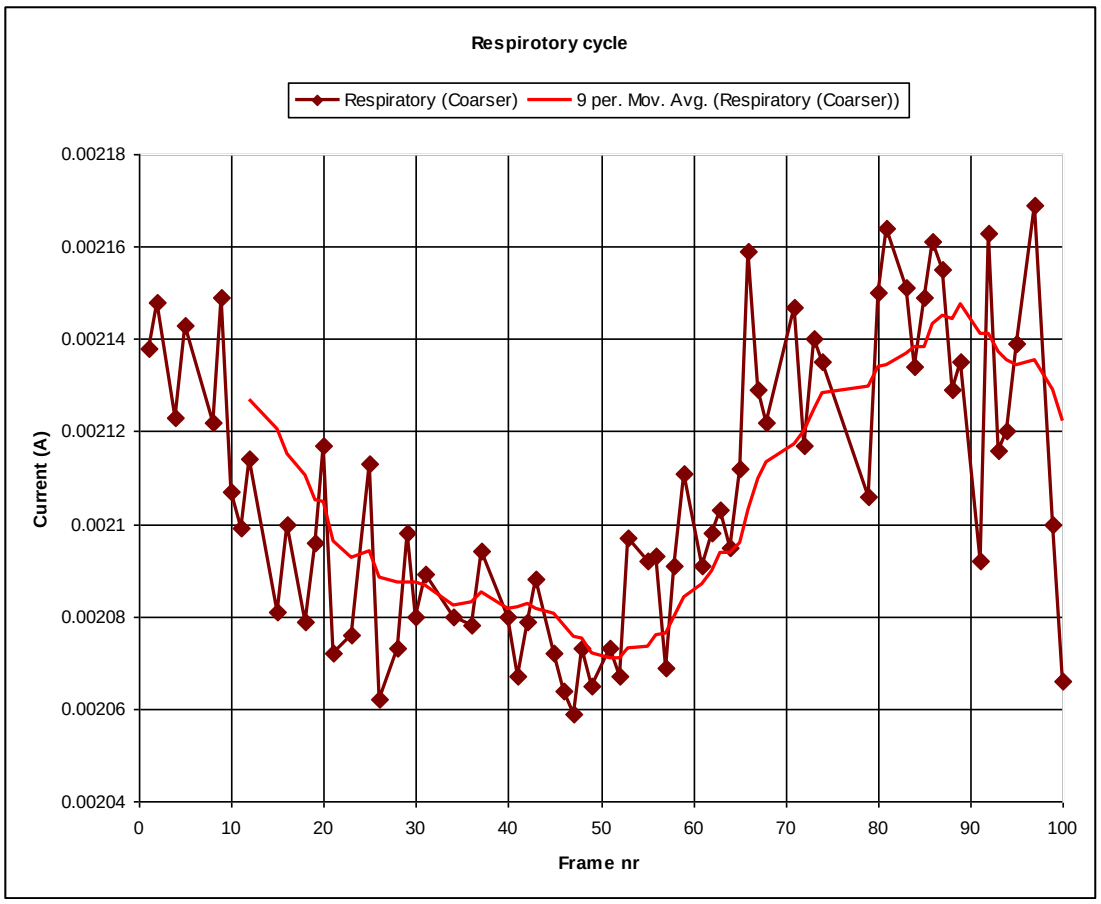

Figure 12. The respiration values' graph for simulation with mesh parameter "coarser". The interval between each two frames is 0.06 seconds.

The respiratory cycle can be explicitly observed on this graph.

While inhaling the lungs are expanding. So the impedance of the thorax is increasing and intensity of a current is decreasing.

On the exhale the lungs are diminishing. So the impedance of the thorax is decreasing and intensity of a current is increasing.

The last part of the graph is the beginning of the new respiratory cycle.

If we change the parameter for mesh from "coarser" to "coarse", only 53 values from 100 will be left. But it is still enough to restore the signal.

The graphs for both simulations:

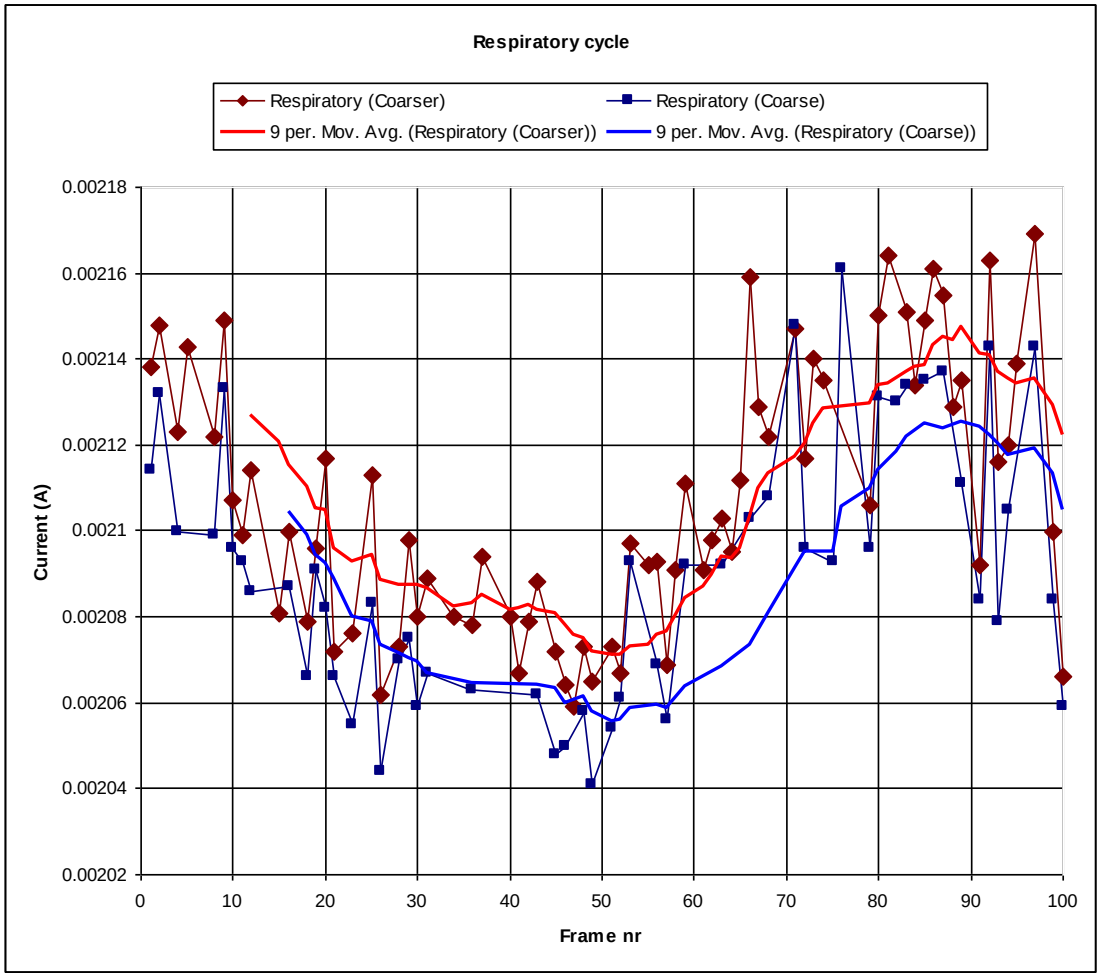

Figure 13. The respiration values' graph for simulations with mesh parameters "coarser" and "coarse". The interval between each two frames is 0.06 seconds.

Then the heart was also added to the simulated project. The obvious way was to use the pericardium for simulation. However, the volume changes of the pericardium during the heartbeat are too small to influence the measurements. The most changeable parts of the heart are atriums and ventricles of the heart.

In addition, pericardium surface is located in initial data very close to the lungs surface. The simplification and fixing slightly change the surface points' position so that pericardium and lungs are intersecting. The intersection causes the COMSOL simulation failure very often.

Due to the foregoing reasons only the inner parts of the heart were used for simulation. The geometry used for the simulation:

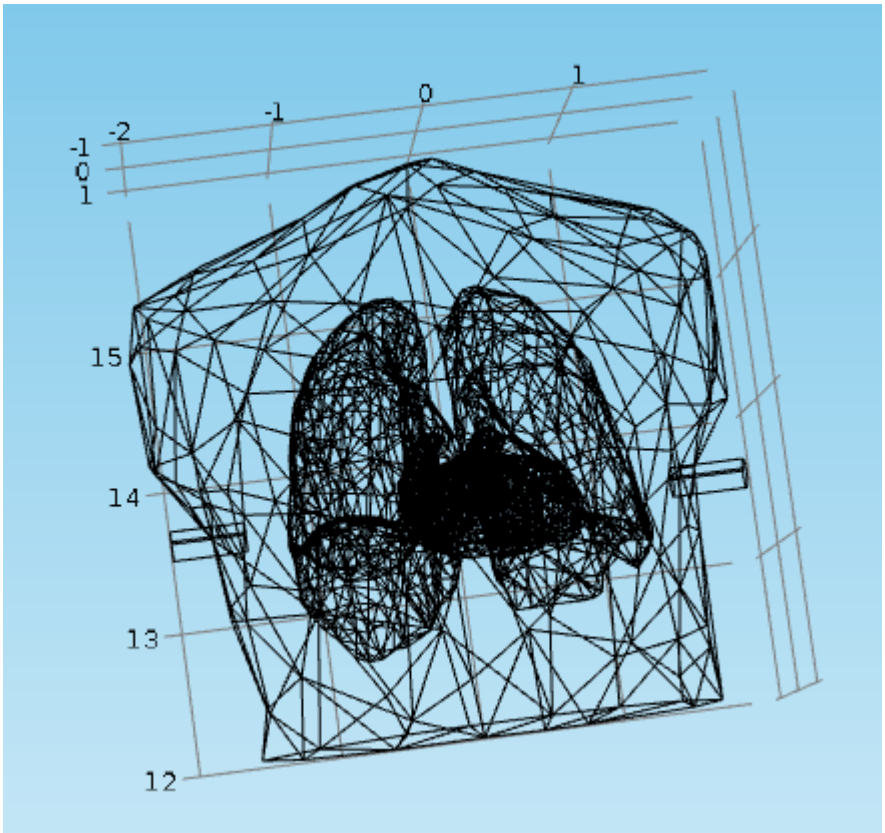

Figure 14. The surfaces of electrodes, lungs, thorax, left and right ventricles of the heart and left and right atriums

The number of succeeded frames decreased to 28 of 100. It is not enough to reconstruct the signal.

The graph with respiratory and heartbeat simulation:

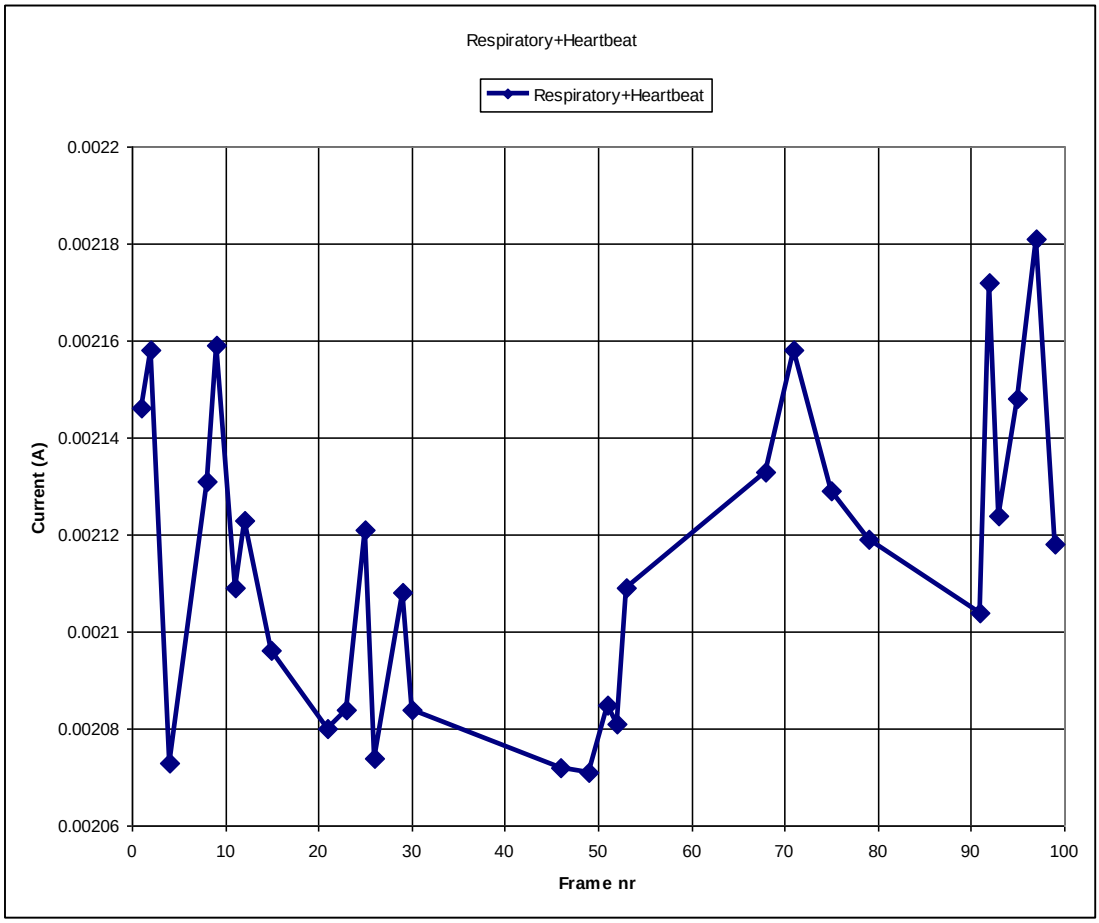

Figure 15. The respiration and heartbeat values' graph

So the third objective has been partly attained. The developed system works well with respiratory cycle, but it is not applicable for detecting the heartbeat.

6 Conclusion and future work

During the present research the majority of the objectives has been successfully gained.

It has been found how to save the generated by 4D XCAT data as the collection of surface points' coordinates. The points are saved into the RAW file which can be converted into the STL file. The STL files can be imported to the COMSOL so the second objective has also been aimed. The third objective was only partly attained. It is possible to simulate the lungs' movements and to restore the respiratory signal with present modelling system. However, for heartbeat signal measurement simulation it is not stable enough. COMSOL is not able to mesh the body and this might happen because of intersecting domain surfaces. Some COMSOL problems also occur when trying to build the geometry. It might be supposed that certain of the surfaces intersect at some disadvantageous angle.

There are some future improvements that should make the system more precise and stable:

- The system should be improved and made more stable to receive more values from the signal. That allows to simulate and to restore also the heartbeat signal. Maybe the solution is to write the script that moves the different surfaces apart. Than there will be less intersected surfaces in the projects and it will be easier to simulate it.
- The system can be simulated in different frequencies. It will help to find the optimal frequency for specific applications.
- More electrodes can be added to the model and best electrode locations can be found for best sensitivity for specific applications.

Appendixes

Appendix 1. MATLAB script "build_data_stl"

Appendix 2. MATLAB script "make_stl"

Appendix 3. MATLAB script "build_data_comsol"

Appendix 4. MATLAB script "config"

Appendix 5. MATLAB script "readbin.m"

Appendix 6. MeshLab filters' file "default.mlx"

Appendix 7. Command line file "run.cmd"

Appendix 8. MATLAB script "isosurf.m"

Appendix 9. MATLAB script "stlwrite.m"

Appendix 10. MATLAB script "surf2stl.m"

Appendix 11. MATLAB script "bodybuilder.m"

Appendix 12. Tissues' properties

Appendix 1. MATLAB script "build_data_stl" - Makes STL files for all given frames.

```
function  build_data_stl
    global cfg_frame_start;
    global cfg_frame_end;

    config;

    % make STLs for all frames
    for frame_num = cfg_frame_start:cfg_frame_end
        make_frame_stls(frame_num);
    end
    fclose('all');
end


function make_frame_stls(frame_num)
    global cfg_body_part;
    for i = 1:numel(cfg_body_part)
        make_stl(frame_num, char(cfg_body_part(i)));
    end
end
```

Appendix 2. MATLAB script "make_stl" - Makes STL files from given RAW data for one frame.

```
function  make_stl( frame_num, body_part )
    global cfg_root_path;
    global cfg_zmin;
    global cfg_zmax;

    fprintf('processing %s #%d\n', body_part, frame_num);
    infile  =  sprintf(  '%s\\data_xcat\\data_%d_%s.raw',  cfg_root_path,  frame_num,
body_part);

    % open input file
    fin = fopen(infile,'rt');
    if (fin == -1)
        error( sprintf('Unable to open %s',infile) );
    end

    fout = -1;

    facets_cnt = 0;

    last_title = '';

    tline = fgets(fin);
    while ischar(tline)
        tline = tline(1:numel(tline)-1);
        if isempty(tline)
            tline = fgets(fin);
            continue;
        end
        values = sscanf(tline,'%f %f %f %f %f %f %f %f %f');
        if numel(values) == 9
            % writing stl triangle
            %values = floor(values*10)/1000;
            values = round(values*10)/1000;
%           values = values/100;
            p1 = [values(1), values(2), values(3)];
            p2 = [values(4), values(5), values(6)];
            p3 = [values(7), values(8), values(9)];

            zmin_intersection = zcompare(p1,p2,p3, cfg_zmin);
            zmax_intersection = zcompare(p1,p2,p3, cfg_zmax);

            if zmin_intersection == 0
                if p1(3) <= cfg_zmin
                    p1(3) = cfg_zmin;
                    bottom_planar = [bottom_planar; p1];
```

```
                end
                if p2(3) <= cfg_zmin
                    p2(3) = cfg_zmin;
                    bottom_planar = [bottom_planar; p2];
                end
                if p3(3) <= cfg_zmin
                    p3(3) = cfg_zmin;
                    bottom_planar = [bottom_planar; p3];
                end
            end

            if zmax_intersection == 0
                if p1(3) >= cfg_zmax
                    p1(3) = cfg_zmax;
                    top_planar = [top_planar; p1];
                end
                if p2(3) >= cfg_zmax
                    p2(3) = cfg_zmax;
                    top_planar = [top_planar; p2];
                end
                if p3(3) >= cfg_zmax
                    p3(3) = cfg_zmax;
                    top_planar = [top_planar; p3];
                end
            end

            if zmin_intersection >= 0 && zmax_intersection <= 0
                local_write_facet(fout, p1, p2, p3);
                facets_cnt = facets_cnt + 1;
            end

%             if zmin_intersection == 0
%                 bottom_planar = [bottom_planar; bottom_points(p1,p2,p3)];
%             end
%             if zmax_intersection == 0
%                 top_planar = [top_planar; top_points(p1,p2,p3)];
%             end

        else
            if strcmp(tline, last_title) % merge to single stl
                tline = fgets(fin);
                continue;
            end
            last_title = tline;
            % new header read
            % 1 close prev. file if any
            if fout ~= -1

                % enclose top cut
                top_planar = unique(top_planar,'rows');
                if size(top_planar,1) > 2
                    planar = build_surface(top_planar);
                    fprintf('top cut %d ', size(planar,1));
                    for i = 1:3:size(planar,1);
                        local_write_facet(fout,       planar(i,:),       planar(i+1,:),
planar(i+2,:));
                        facets_cnt = facets_cnt + 1;
                    end
                end

                % enclose bottom cut
                bottom_planar = unique(bottom_planar,'rows');
                if size(bottom_planar,1) > 2
                    planar = build_surface(bottom_planar);
                    fprintf('bottom cut %d ', size(planar,1));
                    for i = 1:3:size(planar,1);
                        local_write_facet(fout,       planar(i,:),       planar(i+1,:),
planar(i+2,:));
                        facets_cnt = facets_cnt + 1;
                    end
                end

                fseek(fout,80,'bof');
                fwrite(fout,facets_cnt,'int32');
                fclose(fout);
```

```
                fout = -1;
                fprintf('%d facets\n', facets_cnt);
                if facets_cnt == 0
                    delete(outfile);
                end
            end
            % 2 open a new file
            outfile    =    sprintf(    '%s\\data_stl\\%03d_%s_%s.stl',    cfg_root_path,
frame_num, body_part, tline);
            fout = fopen(outfile,'wb+');
            if (fout == -1)
                error( sprintf('Unable to open %s',outfile) );
            end
            %write header to the new file
            fwrite(fout,sprintf('%-80s',tline),'uchar');     % Title
            fwrite(fout,0,'int32');  % Number of facets, zero for now
            facets_cnt = 0;
            top_planar = zeros(0);
            bottom_planar = zeros(0);
            fprintf('\t%s ... ', tline);
        end
        tline = fgets(fin);
    end

    if fout ~= -1
        fseek(fout,80,'bof');
        fwrite(fout,facets_cnt,'int32');
        fclose(fout);
        fout = -1;
        fprintf('%d facets\n', facets_cnt);
        if facets_cnt == 0
            delete(outfile);
        end
    end

    fclose(fin);
end

% Local subfunction
function local_write_facet(fid,p1,p2,p3)
    n = local_find_normal(p1,p2,p3);
    fwrite(fid,n,'float32');
    fwrite(fid,p1,'float32');
    fwrite(fid,p2,'float32');
    fwrite(fid,p3,'float32');
    fwrite(fid,0,'int16');  % unused
end

function n = local_find_normal(p1,p2,p3)
    v1 = p2-p1;
    v2 = p3-p1;
    v3 = cross(v1,v2);
    n = v3 ./ sqrt(sum(v3.*v3));
end

% compares triangle p1, p2, p3 with planar z
% returns:
% -1 -- triangle < z
% 0 -- triangle intersects z
% 1 -- triangle > z
function r = zcompare(p1,p2,p3, z)
    zets = [p1(3), p2(3), p3(3)];
    r = 0;
    if all(zets < z) == 1
        r = -1;
    else
        if all(zets >= z) == 1
            r = 1;
        end
    end
end

% returns 2 top points of triangle p1,p2,p3
function r = top_points(p1,p2,p3)
    points = [p1;p2;p3];
```

```
    [~,i] = max(points(:,3));
    r = points(i,:);
end

% returns 2 bottom points of triangle p1,p2,p3
function r = bottom_points(p1,p2,p3)
    points = [p1;p2;p3];
    [~,i] = min(points(:,3));
    r = points(i,:);
end

% builds surface's triangles having set of boundary points
function triangles = build_surface(boundary)
    triangles = zeros(0);

    % find central point
    center = [mean(boundary(:,1)), mean(boundary(:,2))];

    % calculate angles
    angles = zeros(size(boundary,1),1);
    for i = 1:size(boundary,1)
        angles(i) = atan2( boundary(i,2) - center(2), boundary(i,1) - center(1));
    end

    % sort by ange
    [angles, sorted_i] = sort(angles);

    half_size = floor(size(boundary,1)/2) - 1;

    for i1 = 1:half_size
        i2 = size(boundary,1) - i1 + 1;
        p11 = boundary(sorted_i(i1),:);
        p12 = boundary(sorted_i(i1+1),:);
        p21 = boundary(sorted_i(i2),:);
        p22 = boundary(sorted_i(i2-1),:);
        triangles = [triangles;[p12;p11;p21];[p12;p21;p22]];
    end

    if mod(size(boundary,1),2) ~= 0
        i0 = floor(size(boundary,1)/2)+1;
        p0 = boundary(sorted_i(i0),:);
        p11 = boundary(sorted_i(i0-1),:);
        p12 = boundary(sorted_i(i0+1),:);
        triangles = [triangles;[p0;p11;p12]];
    end
end

function triangles = build_surface1(boundary)
    triangles = zeros(0);

    % find central point
  center = [mean(boundary(:,1)), mean(boundary(:,2)), mean(boundary(:,3)) ];

    % calculate angles
    angles = zeros(size(boundary,1),1);
    for i = 1:size(boundary,1)
        angles(i) = atan2( boundary(i,2) - center(2), boundary(i,1) - center(1));
    end

    % sort by ange
    [~, sorted_i] = sort(angles);

    for i = 2:size(boundary,1)
        p1 = boundary(sorted_i(i-1),:);
        p2 = boundary(sorted_i(i),:);
        triangles = [triangles;[p1;p2;center]];
    end
    p2 = boundary(sorted_i(1),:);
    p1 = boundary(sorted_i(size(boundary,1)),:);
    triangles = [triangles;[p1;p2;center]];
end
```

Appendix 3. MATLAB script "build_data_comsol" - Simulates in COMSOL all the given frames.

```
function build_data_comsol
    global cfg_frame_start;
    global cfg_frame_end;
    global cfg_root_path;
    global cfg_results_file_name;
    global cfg_frames_done;

    config;

    fprintf('***** START **********\n');
    results_file = sprintf('%s\\results\\%s', cfg_root_path, cfg_results_file_name);
    resFile = fopen(results_file,'w');

    fprintf(resFile, 'RESULTS %s\r\n', datestr(now));
    fprintf(resFile, 'Frames from %d to %d\r\n', cfg_frame_start, cfg_frame_end);
    fprintf(resFile, '    FRAME   L. CURRENT   R. CURRENT\r\n');


    % make STLs for all frames
    for frame_num = cfg_frame_start:cfg_frame_end
        if not(isempty(find(cfg_frames_done==frame_num,1)))
            continue;
        end
       try
            make_frame_comsol(frame_num, resFile);
       catch
            fprintf('FAILED\n');
       end
    end

    fclose(resFile);
    fclose('all');
    fprintf('***** FINISH **********\n');
end

function make_frame_comsol(frame_num, resFile)
    global cfg_root_path;

    fprintf('FRAME %d\n', frame_num);
    out_file = sprintf('%sdata_comsol\\%03d', cfg_root_path, frame_num);

    import com.comsol.model.*
    import com.comsol.model.util.*

    model = ModelUtil.create('Model');

    % define file generic items
    model.modelNode.create('comp1');
    model.geom.create('geom1', 3);
    model.physics.create('ec', 'ConductiveMedia', 'geom1');
    model.study.create('std1');
    model.study('std1').create('freq', 'Frequency');
    model.study('std1').feature('freq').activate('ec', true);
    model.mesh.create('mesh1', 'geom1');

    % ***** Define Geometry *******

    % imports
    make_imports(frame_num, model);

    % finalize geometry
    model.geom('geom1').feature('fin').set('action', 'union');
    fprintf('save %s\n', out_file);
    mphsave(model,out_file);

    fprintf('Build geometry\n');
    model.geom('geom1').run('fin');

    fprintf('save %s\n', out_file);
    mphsave(model,out_file);
```

```
    % assigning materials
    assign_materials(model);

    % ***** Define Mesh *******
    fprintf('save %s\n', out_file);
    mphsave(model,out_file);

    fprintf('Meshing...\n');
    model.mesh('mesh1').automatic(true);
    model.mesh('mesh1').autoMeshSize(6);
    model.mesh('mesh1').run;

    % connect electrodes
    connect_electrodes(model);

    fprintf('save %s\n', out_file);
    mphsave(model,out_file);

    % study
    make_study(frame_num, model);

    % measure resulting values
    fprintf('Get results...\n');
    valLeft = measure_current(model, 'sel_el_left');
    valRight = measure_current(model, 'sel_el_right');
    fprintf(resFile, '%8d%12.6f%12.6f\r\n', frame_num, valLeft, valRight);

    fprintf('save %s\n', out_file);
    mphsave(model,out_file);

end

function make_material(model, cfg_name, domain_name)
    global cfg_comsol_material;

    for iMat = 1:size(cfg_comsol_material,1);
        if strcmp(cfg_comsol_material{iMat}{1},cfg_name)
            mat    =    model.material.create(domain_material(domain_name),    'Common',
'comp1');
            mat.label( sprintf('%s - %s', domain_name, cfg_name) );
            for iProp = 1:size(cfg_comsol_material{iMat}{2});
                mat.propertyGroup('def').set(cfg_comsol_material{iMat}{2}{iProp}{1},
{cfg_comsol_material{iMat}{2}{iProp}{2}});
            end
            break;
        end
    end
end

function n = domain_selection(domain_name)
    n = sprintf('geom1_sel_%s_dom', domain_name);
end

function n = bnd_selection(domain_name)
    n = sprintf('geom1_sel_%s_bnd', domain_name);
end

function n = domain_material(domain_name)
    n = sprintf('mat_%s', domain_name);
end

% import STLs
function make_imports(frame_num, model)
    global cfg_root_path;
    global cfg_comsol_domain;

    % import
    for i = 1:size(cfg_comsol_domain,1);
        name = cfg_comsol_domain{i}{1};

        if strcmp(name,'cyl_right') || strcmp(name,'cyl_left')
           file  =  sprintf('%sdata_stl\\%s.stl',  cfg_root_path,cfg_comsol_domain{i}{2}
);
        else
```

```
            file        =        sprintf('%sdata_stl\\%03d_%s.stl',        cfg_root_path,
frame_num,cfg_comsol_domain{i}{2} );
        end

        params = cfg_comsol_domain{i}{6};
        cfg_material = cfg_comsol_domain{i}{3};

        fprintf('Importing %s\n', file);

        imp = model.geom('geom1').create(name, 'Import');
        imp.set('filename', file);

        if strcmp(name,'cyl_right') || strcmp(name,'cyl_left')
            imp.set('facepartition', 'auto');
        else
            for iProp = 1:size(params,1);
                imp.set(params{iProp}{1},params{iProp}{2});
            end
        end

        % define selection
        sel_name = sprintf('sel_%s', name);
        sel = model.geom('geom1').selection.create(sel_name, 'CumulativeSelection');
        sel.label(sel_name);
        imp.set('createselection', 'on');
        imp.set('contributeto', sel_name);
        imp.importData;


        model.geom('geom1').run('fin');

        make_material(model, cfg_material, name);
    end


end

function make_electrodes_(model)
    global cfg_comsol_sphere;
    for i = 1:size(cfg_comsol_sphere,1);
        name = cfg_comsol_sphere{i}{1};
        type = cfg_comsol_sphere{i}{2};
        X = cfg_comsol_sphere{i}{3};
        Y = cfg_comsol_sphere{i}{4};
        Z = cfg_comsol_sphere{i}{5};
        R = cfg_comsol_sphere{i}{6};
        params = cfg_comsol_sphere{i}{7};

        %sphere
        sphere = model.geom('geom1').create(name, 'Sphere');
        sphere.label(name);
        sphere.set('pos', {X Y Z});
        sphere.set('r', R);

        % define selection
        sel_name = sprintf('sel_%s', name);
        sel = model.geom('geom1').selection.create(sel_name, 'CumulativeSelection');
        sel.label(sel_name);

        sphere.set('createselection', 'on');
        sphere.set('contributeto', sel_name);
        model.geom('geom1').run(name);

        make_material(model, 'electrode', name);

        model.material.move(domain_material(name), 0);
        model.geom('geom1').feature.move(name, 0);
    end
end

% assign materials to all domains
function assign_materials(model)
    global cfg_comsol_domain;

    % imports
```

```
    for i = 1:size(cfg_comsol_domain,1);
        name = cfg_comsol_domain{i}{1};
        cfg_material = cfg_comsol_domain{i}{3};
        fprintf('Material of %s is %s\n', name, cfg_material);
        model.material(domain_material(name)).selection.named(domain_selection(name));
    end

end

function connect_electrodes(model)
    global cfg_comsol_electrode;
    for i = 1:size(cfg_comsol_electrode,1);
        name = cfg_comsol_electrode{i}{1};
        type = cfg_comsol_electrode{i}{2};
        X = cfg_comsol_electrode{i}{3};
        Y = cfg_comsol_electrode{i}{4};
        Z = cfg_comsol_electrode{i}{5};
        R = cfg_comsol_electrode{i}{6};
        params = cfg_comsol_electrode{i}{7};

        sel_name = sprintf('sel_%s', name);
        sel = model.geom('geom1').create(sel_name, 'BallSelection');
        sel.set('posx', X);
        sel.set('posy', Y);
        sel.set('posz', Z);
        sel.set('r', R);
        sel.set('entitydim', '2');
        sel.set('condition', 'inside');
        sel.label(sel_name);
        model.geom('geom1').run(sel_name);

        % electode
        el = model.physics('ec').feature.create(name, type, 2);
        for iProp = 1:size(params,1);
            el.set(params{iProp}{1},params{iProp}{2});
        end
        el.selection.named(sprintf('geom1_%s', sel_name));
    end
end

function make_study(frame_num, model)
    global cfg_root_path;

    jpg_file = sprintf('%sdata_jpg\\%03d.jpg', cfg_root_path, frame_num);

    fprintf('Running study...\n');
    model.study('std1').feature('freq').set('plist', '10');
    model.sol.create('sol1');
    model.sol('sol1').study('std1');

    model.study('std1').feature('freq').set('notlistsolnum', 1);
    model.study('std1').feature('freq').set('notsolnum', '1');
    model.study('std1').feature('freq').set('listsolnum', 1);
    model.study('std1').feature('freq').set('solnum', '1');

    model.sol('sol1').create('st1', 'StudyStep');
    model.sol('sol1').feature('st1').set('study', 'std1');
    model.sol('sol1').feature('st1').set('studystep', 'freq');
    model.sol('sol1').create('v1', 'Variables');
    model.sol('sol1').feature('v1').set('control', 'freq');
    model.sol('sol1').create('s1', 'Stationary');
    model.sol('sol1').feature('s1').create('p1', 'Parametric');
    model.sol('sol1').feature('s1').feature.remove('pDef');
    model.sol('sol1').feature('s1').feature('p1').set('pname', {'freq'});
    model.sol('sol1').feature('s1').feature('p1').set('plistarr', {'10'});
    model.sol('sol1').feature('s1').feature('p1').set('punit', {'Hz'});
    model.sol('sol1').feature('s1').feature('p1').set('pcontinuationmode', 'no');
    model.sol('sol1').feature('s1').feature('p1').set('preusesol', 'auto');
    model.sol('sol1').feature('s1').feature('p1').set('plot', 'off');
    model.sol('sol1').feature('s1').feature('p1').set('plotgroup', 'Default');
    model.sol('sol1').feature('s1').feature('p1').set('probesel', 'all');
    model.sol('sol1').feature('s1').feature('p1').set('probes', {});
    model.sol('sol1').feature('s1').feature('p1').set('control', 'freq');
    model.sol('sol1').feature('s1').set('control', 'freq');
    model.sol('sol1').feature('s1').create('fc1', 'FullyCoupled');
```

```
    model.sol('sol1').feature('s1').create('i1', 'Iterative');
    model.sol('sol1').feature('s1').feature('i1').set('linsolver', 'bicgstab');
    model.sol('sol1').feature('s1').feature('i1').create('mg1', 'Multigrid');
    model.sol('sol1').feature('s1').feature('fc1').set('linsolver', 'i1');
    model.sol('sol1').feature('s1').feature.remove('fcDef');
    model.sol('sol1').attach('std1');

    % list graph

    model.result.create('pg1', 'PlotGroup3D');
    model.result('pg1').label('Plots');
    model.result('pg1').set('oldanalysistype', 'noneavailable');
    model.result('pg1').set('frametype', 'spatial');
    model.result('pg1').set('data', 'dset1');
    model.result('pg1').feature.create('mslc1', 'Multislice');
    model.result('pg1').feature('mslc1').label('Current Density');
    model.result('pg1').feature('mslc1').set('oldanalysistype', 'noneavailable');
    model.result('pg1').feature('mslc1').set('data', 'parent');
    model.result('pg1').feature('mslc1').set('rangeunit', 'A/m^2');
    model.result('pg1').feature('mslc1').set('descr', 'Current density module');
    model.result('pg1').feature('mslc1').set('expr', 'sqrt(ec.Jx^2+ec.Jy^2+ec.Jx^2)');
    model.result('pg1').feature('mslc1').set('rangecoloractive', 'on');
    model.result('pg1').feature('mslc1').set('rangecolormin', '0.000000001');
    model.result('pg1').feature('mslc1').set('rangecolormax', '0.002');
    model.result('pg1').create('str1', 'Streamline');
    model.result('pg1').feature('str1').set('posmethod', 'start');
    model.result('pg1').feature('str1').set('number', '50');


    % export
    exp = model.result.export.create('img1', 'pg1', 'Image3D');
    exp.set('jpegfilename', jpg_file);

    model.sol('sol1').runAll;

    model.label('001.mph');

    model.result('pg1').set('window', 'graphics');
    model.result('pg1').set('windowtitle', '');
    model.result('pg1').run;

    exp.run;



end

function res = measure_current(model, sel_name)
    val    =    model.result.numerical.create(    sprintf('current_%s',    sel_name),
'IntSurface');
    val.selection.named(sprintf('geom1_%s', sel_name));
    val.set('expr', 'ec.normJ');
    res = val.getReal(1);
end
```

Appendix 4. MATLAB script "config" - Includes all the data needed for other scripts.

```
function config

% ****************************************************
% common parameters
% ****************************************************
global cfg_root_path;
cfg_root_path = 'C:\!My_documents\Studies\diploma\';

global cfg_frame_start;
cfg_frame_start = 1;
global cfg_frame_end;
cfg_frame_end = 100;

global cfg_frames_done;
cfg_frames_done=[];% frames that should not be converted

global cfg_results_file_name;
cfg_results_file_name = 'results.txt';

% ****************************************************
% stl parameters
% ****************************************************
global cfg_body_part;
%cfg_body_part = {'body', 'bones', 'heart', 'muscle', 'organs', 'vessels'};
cfg_body_part = {'body', 'heart', 'organs'};
%cfg_body_part = {'body', 'heart'};
%cfg_body_part = {'heart'};

% cutting planars
global cfg_zmin;
cfg_zmin = 12;
global cfg_zmax;
cfg_zmax = 17;


% ****************************************************
% comsol parameters
% ****************************************************
% material: name, { name, value }
% domain: name, stl_file, { name, value }, material, mesh_feature_flat,
%           mesh_feature, min_flat_size

global cfg_comsol_material;
cfg_comsol_material = { ...
{ 'electrode'; ...
    { ...
        { 'electricconductivity', '5.998e1' }; ...
        { 'relpermittivity', '1' } ...
    } };...
{ 'muscle'; ...
    { ...
        { 'electricconductivity', '0.20197' }; ...
        { 'relpermittivity', '25700000' } ...
    } };...
{ 'heart'; ...
    { ...
        { 'electricconductivity', '0.053677' }; ...
        { 'relpermittivity', '23562000' } ...
    } }; ...
{ 'lung'; ...
    { ...
        { 'electricconductivity', '0.038904' }; ...
        { 'relpermittivity', '32248000' } ...
    } }; ...
{ 'blood'; ...
    { ...
        { 'electricconductivity', '0.7' }; ...
        { 'relpermittivity', '5260' } ...
    } } ...
};

%  name,  stl_sufix,  material,  min_flat_mesh,  flat_mesh,  mesh,  import_params{name,
value}
```

```
global cfg_comsol_domain;
cfg_comsol_domain = { ...
{ 'cyl_right'; 'cyl_right'; 'electrode'; 1; 'FreeTet'; ...
    { ...
        { 'facepartition', 'auto' }; ...
        { 'faceangle', '0' }; ...
        { 'neighangle', '0' }; ...
        { 'planar', 'off' }; ...
        { 'facecleanup', '0' } ...
    } };...

{ 'cyl_left'; 'cyl_left'; 'electrode'; 1; 'FreeTet'; ...
    { ...
        { 'facepartition', 'auto' }; ...
        { 'faceangle', '0' }; ...
        { 'neighangle', '0' }; ...
        { 'planar', 'off' }; ...
        { 'facecleanup', '0' } ...
    } };...

{ 'body'; 'body_chest_surface_fixed'; 'muscle'; 1; 'FreeTet'; ...
    { ...
        { 'facepartition', 'manual' }; ...
        { 'faceangle', '0' }; ...
        { 'neighangle', '0' }; ...
        { 'planar', 'off' }; ...
        { 'facecleanup', '0' } ...
    } };...


{ 'rlung'; 'organs_rlung_fixed'; 'lung'; 1; 'FreeTet'; ...
    { ...
        { 'facepartition', 'manual' }; ...
        { 'faceangle', '0' }; ...
        { 'neighangle', '0' }; ...
        { 'planar', 'off' }; ...
        { 'facecleanup', '0' } ...
    } };...
{ 'llung'; 'organs_llung_fixed'; 'lung'; 1; 'FreeTet'; ...
    { ...
        { 'facepartition', 'manual' }; ...
        { 'faceangle', '0' }; ...
        { 'neighangle', '0' }; ...
        { 'planar', 'off' }; ...
        { 'facecleanup', '0' } ...
    } }...
%{ 'heart'; 'heart_dias_pericardium_fixed'; 'heart'; 1; 'FreeTet'; ...
%    { ...
%        { 'facepartition', 'manual' }; ...
%        { 'faceangle', '0' }; ...
%        { 'neighangle', '0' }; ...
%        { 'planar', 'off' }; ...
%        { 'facecleanup', '0' } ...
%    } };...
%{ 'heart_la'; 'heart_dias_la_fixed'; 'blood'; 1; 'FreeTet'; ...
%   { ...
%      { 'facepartition', 'manual' }; ...
%       { 'faceangle', '0' }; ...
%      { 'neighangle', '0' }; ...
%     { 'planar', 'off' }; ...
%    { 'facecleanup', '0' } ...
%    } };...
%{ 'heart_lv'; 'heart_dias_lv_0_fixed'; 'blood'; 1; 'FreeTet'; ...
%    { ...
%        { 'facepartition', 'manual' }; ...
%        { 'faceangle', '0' }; ...
%        { 'neighangle', '0' }; ...
%        { 'planar', 'off' }; ...
%        { 'facecleanup', '0' } ...
%    } };...
%{ 'heart_ra'; 'heart_dias_ra_fixed'; 'blood'; 1; 'FreeTet'; ...
%    { ...
%        { 'facepartition', 'manual' }; ...
%        { 'faceangle', '0' }; ...
%        { 'neighangle', '0' }; ...
```

```
%        { 'planar', 'off' }; ...
%       { 'facecleanup', '0' } ...
%    } };...
%{ 'heart_rv'; 'heart_dias_rv_fixed'; 'blood'; 1; 'FreeTet'; ...
%    { ...
%        { 'facepartition', 'manual' }; ...
%        { 'faceangle', '0' }; ...
%        { 'neighangle', '0' }; ...
%        { 'planar', 'off' }; ...
%        { 'facecleanup', '0' } ...
%    } }...

};

% electrodes
% each electrode connected horizontally left or right side
%  electrode:  name,  type(Ground,  ElectricPotential),  ballX,  ballY,  ballZ,  ballR,
params{name,value}
global cfg_comsol_electrode;
cfg_comsol_electrode = { ...
{ 'el_left'; 'Ground'; '-2'; '0'; '13.5'; '.15';...
    { ...
    } };...
{ 'el_right'; 'ElectricPotential'; '1.9'; '0'; '13.5'; '.15'; ...
    { ...
        { 'V0', '0.05' }; ...
    } }...
};

global cfg_comsol_sphere;
cfg_comsol_sphere = { ...
{ 'sphere_left'; 'Ground'; '-1.6'; '0'; '13'; '.3';...
    { ...
    } };...
{ 'sphere_right'; 'ElectricPotential'; '1.5'; '0'; '13'; '.3'; ...
    { ...
        { 'V0', '1' }; ...
    } }...
};
```

Appendix 5. MATLAB script "readbin.m" - Makes the STL file from 3D matrix data.

```
% possible materials
%       0
%       2
%       4
%       5
%       6
%      30
%      50
%      60
%      75

points = multibandread('C:\!My_documents\Studing\XCAD\result_act_2.bin', [256, 256, 500], 'float',0, 'bsq','ieee-le' );
Xmax = 256;
Ymax = 256;
Zmax = 500;

targetMaterial0 = 60;
%targetMaterial1 = 6;

outZ = zeros([Xmax Ymax,16]);
surfaceCnt= 0;

lastSeen = 0;
disp('Finding Surface...');
for x = 1:Xmax
    for y = 1:Ymax
        lastSeen = points(x,y,1);
        borderIdx = 0;
        for z = 1:Zmax
            if points(x,y,z) ~= targetMaterial0%< targetMaterial0 || points(x,y,z) > targetMaterial1
                points(x,y,z) = 0;
            end
            if points(x,y,z) ~= lastSeen
                borderIdx = borderIdx + 1;
                if surfaceCnt < borderIdx
                    surfaceCnt = borderIdx;
                end
                outZ(x,y, borderIdx) = z;
                lastSeen = points(x,y,z);
            end
        end
    end
end

for iS = 1:surfaceCnt
    fprintf('Make STL #%d', iS);
    surf2stl( sprintf('result%d.stl',iS),1,1,outZ(:,:,iS),'ascii');
end
```

Appendix 6. MeshLab filters' file "default.mlx" - The filters file used by MeshLab when fixing.

```
<!DOCTYPE FilterScript>
<FilterScript>
 <filter name="Remove Duplicated Vertex"/>
 <filter name="Remove Duplicate Faces"/>
 <filter name="Quadric Edge Collapse Decimation">
  <Param tooltip="The desired final number of faces." name="TargetFaceNum" value="0" description="Target number of faces" type="RichInt"/>
  <Param tooltip="If non zero, this parameter specifies the desired final size of the mesh as a percentage of the initial size." name="TargetPerc" value="0.1" description="Percentage reduction (0..1)" type="RichFloat"/>
  <Param tooltip="Quality threshold for penalizing bad shaped faces.&lt;br>The value is in the range [0..1]&#xa; 0 accept any kind of face (no penalties),&#xa; 0.5  penalize faces with quality &lt; 0.5, proportionally to their shape&#xa;" name="QualityThr" value="0.5" description="Quality threshold" type="RichFloat"/>
  <Param tooltip="The simplification process tries to do not affect mesh boundaries during simplification" name="PreserveBoundary" value="true" description="Preserve Boundary of the mesh" type="RichBool"/>
  <Param tooltip="The importance of the boundary during simplification. Default (1.0) means that the boundary has the same importance of the rest. Values greater than 1.0 raise boundary importance and has the effect of removing less vertices on the border. Admitted range of values (0,+inf). " name="BoundaryWeight" value="1" description="Boundary Preserving Weight" type="RichFloat"/>
  <Param tooltip="Try to avoid face flipping effects and try to preserve the original orientation of the surface" name="PreserveNormal" value="false" description="Preserve Normal" type="RichBool"/>
  <Param tooltip="Avoid all the collapses that should cause a topology change in the mesh (like closing holes, squeezing handles, etc). If checked the genus of the mesh should stay unchanged." name="PreserveTopology" value="false" description="Preserve Topology" type="RichBool"/>
  <Param tooltip="Each collapsed vertex is placed in the position minimizing the quadric error.&#xa; It can fail (creating bad spikes) in case of very flat areas. &#xa;If disabled edges are collapsed onto one of the two original vertices and the final mesh is composed by a subset of the original vertices. " name="OptimalPlacement" value="true" description="Optimal position of simplified vertices" type="RichBool"/>
  <Param tooltip="Add additional simplification constraints that improves the quality of the simplification of the planar portion of the mesh." name="PlanarQuadric" value="true" description="Planar Simplification" type="RichBool"/>
  <Param tooltip="Use the Per-Vertex quality as a weighting factor for the simplification. The weight is used as a error amplification value, so a vertex with a high quality value will not be simplified and a portion of the mesh with low quality values will be aggressively simplified." name="QualityWeight" value="false" description="Weighted Simplification" type="RichBool"/>
  <Param tooltip="After the simplification an additional set of steps is performed to clean the mesh (unreferenced vertices, bad faces, etc)" name="AutoClean" value="true" description="Post-simplification cleaning" type="RichBool"/>
  <Param tooltip="The simplification is applied only to the selected set of faces.&#xa; Take care of the target number of faces!" name="Selected" value="false" description="Simplify only selected faces" type="RichBool"/>
 </filter>
 <filter name="Select Self Intersecting Faces"/>
 <filter name="Delete Selected Faces"/>
 <filter name="Select non Manifold Edges "/>
 <filter name="Delete Selected Faces"/>
 <filter name="Select non Manifold Vertices"/>
 <filter name="Delete Selected Faces"/>
 <filter name="Close Holes">
  <Param tooltip="The size is expressed as number of edges composing the hole boundary" name="MaxHoleSize" value="30" description="Max size to be closed " type="RichInt"/>
  <Param tooltip="Only the holes with at least one of the boundary faces selected are closed" name="Selected" value="false" description="Close holes with selected faces" type="RichBool"/>
  <Param tooltip="After closing a hole the faces that have been created are left selected. Any previous selection is lost. Useful for example for smoothing the newly created holes." name="NewFaceSelected" value="true" description="Select the newly created faces" type="RichBool"/>
  <Param tooltip="When closing an holes it tries to prevent the creation of faces that intersect faces adjacent to the boundary of the hole. It is an heuristic, non intersetcting hole filling can be NP-complete." name="SelfIntersection" value="true" description="Prevent creation of selfIntersecting faces" type="RichBool"/>
 </filter>
</FilterScript>
```

Appendix 7. command line file "run.cmd" - Runs the simplification and fixing by MeshLab.

```
@echo off
set MESHLAB="C:\Program Files\VCG\MeshLab\meshlabserver.exe"
SET LOG=log.txt
if exist %LOG% (
	del %LOG%
)

rem call :FIX_DOMAIN_FRAMES body_chest_surface
rem call :FIX_DOMAIN_FRAMES heart_dias_pericardium
rem call :FIX_DOMAIN_FRAMES heart_dias_la
rem call :FIX_DOMAIN_FRAMES heart_dias_rv
rem call :FIX_DOMAIN_FRAMES heart_dias_lv_0
rem call :FIX_DOMAIN_FRAMES heart_dias_ra
call :FIX_DOMAIN_FRAMES organs_rlung
call :FIX_DOMAIN_FRAMES organs_llung

goto :EOF


:FIX_DOMAIN_FRAMES
for %%i in (..\data_stl\???_%1.stl) do call :FIX_DOMAIN %1 %%i
goto :EOF

:FIX_DOMAIN
SET MLX=%1.mlx
SET INP=%2
SET OUT=%~dpn2_fixed.stl

if not exist %MLX% (
	SET MLX=default.mlx
)

if not exist %MLX% (
	echo ERROR: %INP%: MLX not found
	goto :EOF
)

echo Processing: %INP% witn %MLX%
%MESHLAB% -i %INP% -o %OUT% -s %MLX% 2>&1 >>%LOG%
```

Appendix 8. MATLAB script "isosurf.m" - Makes the STL file from 3D matrix data.

```
% materials
%       0
%       2
%       4
%       5
%       6
%      30
%      50
%      60
%      75


function make_stl(infile,outfile)
points = multibandread(infile, [256, 256, 500], 'float',0, 'bsq','ieee-le' );
Xmax = 256;
Ymax = 256;
Zmax = 500;

targetMaterial0 = 60;

x_coord = 1:Xmax;
y_coord = 1:Ymax;
z_coord = 1:Zmax;


disp('Finding Surface...');

        surface = isosurface(x_coord, y_coord, z_coord, points, targetMaterial0);
        stlwrite(outfile,surface); % Save to binary .stl

end
```

Appendix 9. MATLAB script "stlwrite.m" - Writes STL file from patch or surface data, script written by Sven Holcombe.

```
function stlwrite(filename, varargin)
%STLWRITE   Write STL file from patch or surface data.
%
%   STLWRITE(FILE, FV) writes a stereolithography (STL) file to FILE for a
%   triangulated patch defined by FV (a structure with fields 'vertices'
%   and 'faces').
%
%   STLWRITE(FILE, FACES, VERTICES) takes faces and vertices separately,
%   rather than in an FV struct
%
%   STLWRITE(FILE, X, Y, Z) creates an STL file from surface data in X, Y,
%   and Z. STLWRITE triangulates this gridded data into a triangulated
%   surface using triangulation options specified below. X, Y and Z can be
%   two-dimensional arrays with the same size. If X and Y are vectors with
%   length equal to SIZE(Z,2) and SIZE(Z,1), respectively, they are passed
%   through MESHGRID to create gridded data. If X or Y are scalar values,
%   they are used to specify the X and Y spacing between grid points.
%
%   STLWRITE(...,'PropertyName',VALUE,'PropertyName',VALUE,...) writes an
%   STL file using the following property values:
%
%   MODE          - File is written using 'binary' (default) or 'ascii'.
%
%   TITLE         - Header text (max 80 chars) written to the STL file.
%
%   TRIANGULATION - When used with gridded data, TRIANGULATION is either:
%                       'delaunay'  - (default) Delaunay triangulation of X, Y
%                       'f'         - Forward slash division of grid quads
%                       'b'         - Back slash division of quadrilaterals
%                       'x'         - Cross division of quadrilaterals
%                   Note that 'f', 'b', or 't' triangulations now use an
%                   inbuilt version of FEX entry 28327, "mesh2tri".
%
%   FACECOLOR     - Single colour (1-by-3) or one-colour-per-face (N-by-3)
%                   vector of RGB colours, for face/vertex input. RGB range
%                   is 5 bits (0:31), stored in VisCAM/SolidView format
%
(http://en.wikipedia.org/wiki/STL_(file_format)#Color_in_binary_STL)
%
%   Example 1:
%       % Write binary STL from face/vertex data
%       tmpvol = zeros(20,20,20);       % Empty voxel volume
%       tmpvol(8:12,8:12,5:15) = 1;     % Turn some voxels on
%       fv = isosurface(tmpvol, 0.99);  % Create the patch object
%       stlwrite('test.stl',fv)         % Save to binary .stl
%
%   Example 2:
%       % Write ascii STL from gridded data
%       [X,Y] = deal(1:40);             % Create grid reference
%       Z = peaks(40);                  % Create grid height
%       stlwrite('test.stl',X,Y,Z,'mode','ascii')

%   Original idea adapted from surf2stl by Bill McDonald. Huge speed
%   improvements implemented by Oliver Woodford. Non-Delaunay triangulation
%   of quadrilateral surface courtesy of Kevin Moerman. FaceColor
%   implementation by Grant Lohsen.
%
%   Author: Sven Holcombe, 11-24-11


% Check valid filename path
path = fileparts(filename);
if ~isempty(path) && ~exist(path,'dir')
    error('Directory "%s" does not exist.',path);
end

% Get faces, vertices, and user-defined options for writing
[faces, vertices, options] = parseInputs(varargin{:});
asciiMode = strcmp( options.mode ,'ascii');

% Create the facets
```

```
facets = single(vertices');
facets = reshape(facets(:,faces'), 3, 3, []);

% Compute their normals
V1 = squeeze(facets(:,2,:) - facets(:,1,:));
V2 = squeeze(facets(:,3,:) - facets(:,1,:));
normals = V1([2 3 1],:) .* V2([3 1 2],:) - V2([2 3 1],:) .* V1([3 1 2],:);
clear V1 V2
normals = bsxfun(@times, normals, 1 ./ sqrt(sum(normals .* normals, 1)));
facets = cat(2, reshape(normals, 3, 1, []), facets);
clear normals

% Open the file for writing
permissions = {'w','wb+'};
fid = fopen(filename, permissions{asciiMode+1});
if (fid == -1)
    error('stlwrite:cannotWriteFile', 'Unable to write to %s', filename);
end

% Write the file contents
if asciiMode
    % Write HEADER
    fprintf(fid,'solid %s\r\n',options.title);
    % Write DATA
    fprintf(fid,[...
        'facet normal %.7E %.7E %.7E\r\n' ...
        'outer loop\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'endloop\r\n' ...
        'endfacet\r\n'], facets);
    % Write FOOTER
    fprintf(fid,'endsolid %s\r\n',options.title);

else % BINARY
    % Write HEADER
    fprintf(fid, '%-80s', options.title);             % Title
    fwrite(fid, size(facets, 3), 'uint32');           % Number of facets
    % Write DATA
    % Add one uint16(0) to the end of each facet using a typecasting trick
    facets = reshape(typecast(facets(:), 'uint16'), 12*2, []);
    % Set the last bit to 0 (default) or supplied RGB
    facets(end+1,:) = options.facecolor;
    fwrite(fid, facets, 'uint16');
end

% Close the file
fclose(fid);
fprintf('Wrote %d facets\n',size(facets, 3));


%% Input handling subfunctions
function [faces, vertices, options] = parseInputs(varargin)
% Determine input type
if isstruct(varargin{1}) % stlwrite('file', FVstruct, ...)
    if ~all(isfield(varargin{1},{'vertices','faces'}))
        error( 'Variable p must be a faces/vertices structure' );
    end
    faces = varargin{1}.faces;
    vertices = varargin{1}.vertices;
    options = parseOptions(varargin{2:end});

elseif isnumeric(varargin{1})
    firstNumInput = cellfun(@isnumeric,varargin);
    firstNumInput(find(~firstNumInput,1):end) = 0; % Only consider numerical input
PRIOR to the first non-numeric
    numericInputCnt = nnz(firstNumInput);

    options = parseOptions(varargin{numericInputCnt+1:end});
    switch numericInputCnt
        case 3 % stlwrite('file', X, Y, Z, ...)
            % Extract the matrix Z
            Z = varargin{3};
```

```
            % Convert scalar XY to vectors
            ZsizeXY = fliplr(size(Z));
            for i = 1:2
                if isscalar(varargin{i})
                    varargin{i} = (0:ZsizeXY(i)-1) * varargin{i};
                end
            end

            % Extract X and Y
            if isequal(size(Z), size(varargin{1}), size(varargin{2}))
                % X,Y,Z were all provided as matrices
                [X,Y] = varargin{1:2};
            elseif numel(varargin{1})==ZsizeXY(1) && numel(varargin{2})==ZsizeXY(2)
                % Convert vector XY to meshgrid
                [X,Y] = meshgrid(varargin{1}, varargin{2});
            else
                error('stlwrite:badinput', 'Unable to resolve X and Y variables');
            end

            % Convert to faces/vertices
            if strcmp(options.triangulation,'delaunay')
                faces = delaunay(X,Y);
                vertices = [X(:) Y(:) Z(:)];
            else
                if ~exist('mesh2tri','file')
                    error('stlwrite:missing', '"mesh2tri" is required to convert X,Y,Z
matrices to STL. It can be downloaded from:\n%s\n',...
                        'http://www.mathworks.com/MATLABcentral/fileexchange/28327')
                end
                [faces, vertices] = mesh2tri(X, Y, Z, options.triangulation);
            end

        case 2 % stlwrite('file', FACES, VERTICES, ...)
            faces = varargin{1};
            vertices = varargin{2};

        otherwise
            error('stlwrite:badinput', 'Unable to resolve input types.');
    end
end

if ~isempty(options.facecolor) % Handle colour preparation
    facecolor = uint16(options.facecolor);
    %Set the Valid Color bit (bit 15)
    c0 = bitshift(ones(size(faces,1),1,'uint16'),15);
    %Red color (10:15), Blue color (5:9), Green color (0:4)
    c0 = bitor(bitshift(bitand(2^6-1, facecolor(:,1)),10),c0);
    c0 = bitor(bitshift(bitand(2^11-1, facecolor(:,2)),5),c0);
    c0 = bitor(bitand(2^6-1, facecolor(:,3)),c0);
    options.facecolor = c0;
else
    options.facecolor = 0;
end

function options = parseOptions(varargin)
IP = inputParser;
IP.addParamValue('mode', 'binary', @ischar)
IP.addParamValue('title', sprintf('Created by stlwrite.m %s',datestr(now)), @ischar);
IP.addParamValue('triangulation', 'delaunay', @ischar);
IP.addParamValue('facecolor',[], @isnumeric)
IP.parse(varargin{:});
options = IP.Results;

function [F,V]=mesh2tri(X,Y,Z,tri_type)
% function [F,V]=mesh2tri(X,Y,Z,tri_type)
%
% Available from http://www.mathworks.com/MATLABcentral/fileexchange/28327
% Included here for convenience. Many thanks to Kevin Mattheus Moerman
% kevinmoerman@hotmail.com
% 15/07/2010
%---------------------------------------------------------------------

[J,I]=meshgrid(1:1:size(X,2)-1,1:1:size(X,1)-1);

switch tri_type
```

```
    case 'f'%Forward slash
        TRI_I=[I(:),I(:)+1,I(:)+1;  I(:),I(:),I(:)+1];
        TRI_J=[J(:),J(:)+1,J(:);   J(:),J(:)+1,J(:)+1];
        F = sub2ind(size(X),TRI_I,TRI_J);
    case 'b'%Back slash
        TRI_I=[I(:),I(:)+1,I(:);  I(:)+1,I(:)+1,I(:)];
        TRI_J=[J(:)+1,J(:),J(:);   J(:)+1,J(:),J(:)+1];
        F = sub2ind(size(X),TRI_I,TRI_J);
    case 'x'%Cross
        TRI_I=[I(:)+1,I(:);  I(:)+1,I(:)+1;  I(:),I(:)+1;    I(:),I(:)];
        TRI_J=[J(:),J(:);    J(:)+1,J(:);    J(:)+1,J(:)+1;  J(:),J(:)+1];
        IND=((numel(X)+1):numel(X)+prod(size(X)-1))';
        F = sub2ind(size(X),TRI_I,TRI_J);
        F(:,3)=repmat(IND,[4,1]);
        Fe_I=[I(:),I(:)+1,I(:)+1,I(:)]; Fe_J=[J(:),J(:),J(:)+1,J(:)+1];
        Fe = sub2ind(size(X),Fe_I,Fe_J);
        Xe=mean(X(Fe),2); Ye=mean(Y(Fe),2);  Ze=mean(Z(Fe),2);
        X=[X(:);Xe(:)]; Y=[Y(:);Ye(:)]; Z=[Z(:);Ze(:)];
end

V=[X(:),Y(:),Z(:)];
```

Appendix 10. MATLAB script "surf2stl.m" - Writes STL file from surface data, script written by Bill McDonald.

```
function surf2stl(filename,x,y,z,mode)
%SURF2STL   Write STL file from surface data.
%   SURF2STL('filename',X,Y,Z) writes a stereolithography (STL) file
%   for a surface with geometry defined by three matrix arguments, X, Y
%   and Z.  X, Y and Z must be two-dimensional arrays with the same size.
%
%   SURF2STL('filename',x,y,Z), uses two vector arguments replacing
%   the first two matrix arguments, which must have length(x) = n and
%   length(y) = m where [m,n] = size(Z).  Note that x corresponds to
%   the columns of Z and y corresponds to the rows.
%
%   SURF2STL('filename',dx,dy,Z) uses scalar values of dx and dy to
%   specify the x and y spacing between grid points.
%
%   SURF2STL(...,'mode') may be used to specify the output format.
%
%     'binary' - writes in STL binary format (default)
%     'ascii'  - writes in STL ASCII format
%
%   Example:
%
%     surf2stl('test.stl',1,1,peaks);
%
%   See also SURF.
%
%   Author: Bill McDonald, 02-20-04

error(nargchk(4,5,nargin));

if (ischar(filename)==0)
    error( 'Invalid filename');
end

if (nargin < 5)
    mode = 'binary';
elseif (strcmp(mode,'ascii')==0)
    mode = 'binary';
end

if (ndims(z) ~= 2)
    error( 'Variable z must be a 2-dimensional array' );
end

if any( (size(x)~=size(z)) | (size(y)~=size(z)) )

    % size of x or y does not match size of z

    if ( (length(x)==1) & (length(y)==1) )
        % Must be specifying dx and dy, so make vectors
        dx = x;
        dy = y;
        x = ((1:size(z,2))-1)*dx;
        y = ((1:size(z,1))-1)*dy;
    end

    if ( (length(x)==size(z,2)) & (length(y)==size(z,1)) )
        % Must be specifying vectors
        xvec=x;
        yvec=y;
        [x,y]=meshgrid(xvec,yvec);
    else
        error('Unable to resolve x and y variables');
    end

end

if strcmp(mode,'ascii')
    % Open for writing in ascii mode
    fid = fopen(filename,'w');
else
    % Open for writing in binary mode
```

```
    fid = fopen(filename,'wb+');
end

if (fid == -1)
    error( sprintf('Unable to write to %s',filename) );
end

title_str = sprintf('Created by surf2stl.m %s',datestr(now));

if strcmp(mode,'ascii')
    fprintf(fid,'solid %s\r\n',title_str);
else
    str = sprintf('%-80s',title_str);
    fwrite(fid,str,'uchar');          % Title
    fwrite(fid,0,'int32');            % Number of facets, zero for now
end

nfacets = 0;

for i=1:(size(z,1)-1)
    for j=1:(size(z,2)-1)

        p1 = [x(i,j)     y(i,j)     z(i,j)];
        p2 = [x(i,j+1)   y(i,j+1)   z(i,j+1)];
        p3 = [x(i+1,j+1) y(i+1,j+1) z(i+1,j+1)];
        val = local_write_facet(fid,p1,p2,p3,mode);
        nfacets = nfacets + val;

        p1 = [x(i+1,j+1) y(i+1,j+1) z(i+1,j+1)];
        p2 = [x(i+1,j)   y(i+1,j)   z(i+1,j)];
        p3 = [x(i,j)     y(i,j)     z(i,j)];
        val = local_write_facet(fid,p1,p2,p3,mode);
        nfacets = nfacets + val;

    end
end

if strcmp(mode,'ascii')
    fprintf(fid,'endsolid %s\r\n',title_str);
else
    fseek(fid,0,'bof');
    fseek(fid,80,'bof');
    fwrite(fid,nfacets,'int32');
end

fclose(fid);

disp( sprintf('Wrote %d facets',nfacets) );



% Local subfunctions

function num = local_write_facet(fid,p1,p2,p3,mode)
if any( p1(3)==0 | p2(3)==0 | p3(3)==0 )
    num = 0;
    return;
end

if any( isnan(p1) | isnan(p2) | isnan(p3) )
    num = 0;
    return;
else
    num = 1;
    n = local_find_normal(p1,p2,p3);

    if strcmp(mode,'ascii')

        fprintf(fid,'facet normal %.7E %.7E %.7E\r\n', n(1),n(2),n(3) );
        fprintf(fid,'outer loop\r\n');
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p1);
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p2);
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p3);
        fprintf(fid,'endloop\r\n');
        fprintf(fid,'endfacet\r\n');
```

```
    else

        fwrite(fid,n,'float32');
        fwrite(fid,p1,'float32');
        fwrite(fid,p2,'float32');
        fwrite(fid,p3,'float32');
        fwrite(fid,0,'int16');  % unused

    end

end


function n = local_find_normal(p1,p2,p3)

v1 = p2-p1;
v2 = p3-p1;
v3 = cross(v1,v2);
n = v3 ./ sqrt(sum(v3.*v3));
```

Appendix 11. MATLAB script "bodybuilder.m" - Prints out with symbol "@" the given material in a slice of 3D matrix

```
function points = bodybuilder
global Xmax;
global Ymax;
global Zmax;
global maxMaterials;

maxMaterials = 16;
Xmax = 256;
Ymax = 256;
Zmax = 500;

points = read_bin('C:\!My_documents\Studing\XCAD\result_act_2.bin');

[cnt,res] = slice_materials(points(:,:,10));
drawSlice(res(:,:,3));
end

% takes x-y section, returns array[ncurves] of array[nPt,x,y]

function res = read_bin(file)
    global Xmax;
    global Ymax;
    global Zmax;
    global maxMaterials;
    res = multibandread('C:\!My_documents\Studing\XCAD\result_act_2.bin', [Xmax, Ymax,
Zmax], 'int32',0, 'bsq','ieee-be' );
end

% slices zsection onto materials' sections
function [cnt,res] = slice_materials(Zsection)
    global Xmax;
    global Ymax;
    global Zmax;
    global maxMaterials;


    materials = int32(0);
    material_bodies_array = zeros([Xmax, Ymax, maxMaterials], 'int32');

    nMaterials = int32(1);

    current_material_idx = int16(1);
    materials(current_material_idx) = Zsection(1,1);

    for x = 1:Xmax
        for y = 1:Ymax
            material = Zsection(x,y);
            if( material ~= materials(current_material_idx) )
                % new material. Already seen?
                existingMaterialIdx = int16(find(materials==material));
                if isempty(existingMaterialIdx)
                    % new material
                    nMaterials = nMaterials + 1;
                    materials(nMaterials) = material;
                    existingMaterialIdx = nMaterials;
                end
                current_material_idx = existingMaterialIdx;
            end
            material_bodies_array(x, y, current_material_idx) = material;
        end
    end
    res = material_bodies_array;
    cnt = nMaterials;
end

function drawSlice(Zsection)
    global Xmax;
    global Ymax;
    for x = 1:Xmax
        for y = 1:Ymax
            if(Zsection(x,y) == 0)
```

```
                fprintf(' ');
            else
                fprintf('@');
            end
        end
        fprintf('\n');
    end
end
```

Appendix 12. Tissues' properties

ALL TISSUES, SINGLE FREQUENCY
FREQUENCY = 10 Hz

Tissue name	Frequency [Hz]	Conductivity [S/m]	Relative permittivity
Air	1.000e+1	0.000e-1	1.000e+0
Aorta	1.000e+1	2.506e-1	1.000e+7
Bladder	1.000e+1	2.028e-1	5.100e+6
Blood	1.000e+1	7.000e-1	5.260e+3
BloodVessel	1.000e+1	2.506e-1	1.000e+7
BodyFluid	1.000e+1	1.500e+0	9.900e+1
BoneCancellous	1.000e+1	7.556e-2	1.002e+7
BoneCortical	1.000e+1	2.003e-2	5.515e+4
BoneMarrow	1.000e+1	9.755e-4	1.005e+6
BrainGreyMatter	1.000e+1	2.751e-2	4.070e+7
BrainWhiteMatter	1.000e+1	2.766e-2	2.763e+7
BreastFat	1.000e+1	1.547e-2	1.186e+7
Cartilage	1.000e+1	1.611e-1	2.010e+7
Cerebellum	1.000e+1	4.751e-2	4.070e+7
CerebroSpinalFluid	1.000e+1	2.000e+0	1.090e+2
Cervix	1.000e+1	3.022e-1	3.975e+7
Colon	1.000e+1	1.221e-2	3.971e+7
Cornea	1.000e+1	4.111e-1	2.011e+7
Duodenum	1.000e+1	5.111e-1	2.010e+7
Dura	1.000e+1	5.003e-1	5.102e+5
EyeSclera	1.000e+1	5.014e-1	2.603e+6
Fat	1.000e+1	1.221e-2	7.974e+6
GallBladder	1.000e+1	9.000e-1	6.091e+3
GallBladderBile	1.000e+1	1.400e+0	1.200e+2
Gland	1.000e+1	5.111e-1	2.010e+7
Heart	1.000e+1	5.368e-2	2.356e+7
Kidney	1.000e+1	5.441e-2	2.799e+7
Lens	1.000e+1	3.111e-1	2.020e+7
Liver	1.000e+1	2.771e-2	1.506e+7
LungDeflated	1.000e+1	2.028e-1	5.497e+6
LungInflated	1.000e+1	3.890e-2	3.225e+7
Lymph	1.000e+1	5.111e-1	2.010e+7
MucousMembrane	1.000e+1	4.022e-4	5.834e+4
Muscle	1.000e+1	2.020e-1	2.570e+7
Nail	1.000e+1	2.003e-2	5.515e+4
Nerve	1.000e+1	1.713e-2	2.007e+7
Oesophagus	1.000e+1	5.111e-1	2.010e+7
Ovary	1.000e+1	3.111e-1	2.010e+7
Pancreas	1.000e+1	5.111e-1	2.010e+7
Prostate	1.000e+1	4.111e-1	2.011e+7
Retina	1.000e+1	5.014e-1	2.603e+6
SkinDry	1.000e+1	2.000e-4	1.136e+3
SkinWet	1.000e+1	4.022e-4	5.834e+4
SmallIntestine	1.000e+1	5.111e-1	2.051e+7
SpinalCord	1.000e+1	1.713e-2	2.007e+7
Spleen	1.000e+1	3.960e-2	4.330e+7
Stomach	1.000e+1	5.111e-1	2.010e+7
Tendon	1.000e+1	2.509e-1	1.992e+7
Testis	1.000e+1	4.111e-1	2.011e+7
Thymus	1.000e+1	5.111e-1	2.010e+7
Thyroid	1.000e+1	5.111e-1	2.010e+7
Tongue	1.000e+1	2.611e-1	2.010e+7
Tooth	1.000e+1	2.003e-2	5.515e+4
Trachea	1.000e+1	3.003e-1	5.504e+5
Uterus	1.000e+1	2.013e-1	3.514e+7
Vacuum	1.000e+1	0.000e-1	1.000e+0
VitreousHumor	1.000e+1	1.500e+0	9.900e+1

References

1 MathWorks [www]

http://www.mathworks.com/ (06/06/2015)

2 MeshLab [www]

http://meshlab.sourceforge.net/ (06/06/2015)

3 COMSOL [www]

http://www.comsol.com/ (06/06/2015)

4 COMSOL Multiphysics [www]

http://en.wikipedia.org/wiki/COMSOL_Multiphysics (06/06/2015)

5 STL (file format) [www]

http://en.wikipedia.org/wiki/STL_%28file_format%29 (06/06/2015)

6 Command-line interface [www]

http://en.wikipedia.org/wiki/Command-line_interface (06/06/2015)

7 Calculation of the Dielectric Properties of Body Tissues [www]

http://niremf.ifac.cnr.it/tissprop/htmlclie/htmlclie.php (human body tissues) (06/06/2015)

8 Electrocardiography [www]

https://en.wikipedia.org/wiki/Electrocardiography (06/06/2015)

9 Positron emission tomography [www]

https://en.wikipedia.org/wiki/Positron_emission_tomography (06/06/2015)

10 Phonocardiography [www]

http://www.britannica.com/EBchecked/topic/457273/phonocardiography (06/06/2015)

11 Echocardiography [www]

https://en.wikipedia.org/wiki/Echocardiography (06/06/2015)

12 Coronary catheterization [www]

http://en.wikipedia.org/wiki/Coronary_catheterization (06/06/2015)

13 Computational human phantom [www]

http://en.wikipedia.org/wiki/Computational_human_phantom (06/06/2015)

14 The dynamic breathing phantom [www]

http://www.rsdphantoms.com/rt_breathing.htm (06/06/2015)

15 W. P. Segars, G. Sturgeon, S. Mendonca and Jason Grimes, B. M. W. Tsui, (2010), 4D XCAT phantom for multimodality imaging research"

16 Dimbylow, P.J. (1996), "The development of realistic voxel phantoms for electromagnetic field dosimetry," in Proceedings of the Workshop on Voxel Phantom Development, Chilton, U.K.

17 STLView - A Free STL Viewer [www]
http://www.freestlview.com/ (06/06/2015)

18 Introduction to LiveLink for MATLAB [www]
http://nf.nci.org.au/facilities/software/COMSOL/4.3/doc/pdf/llMATLAB/IntroductionToLiveLinkForMATLAB.pdf (06/06/2015)

19 S.Grimnes, O.G. Martinsen. (2000). Bioimpedance & Bioelectricity Basics, ACADEMIC PRESS, ISBN 0-12-303260-1

20 Herman P. Schwan, (1955), Electrical properties of body tissues and impedance plethsmography, University of Pennsylvania

21 Geddes, L.A. and Baker, L.E. (1975), Principles of Applied Biomedical

Instrumentation (2nd Edn), John Wiley and Sons, New York,

22 Nyboer, J. (1959), Electrical Impedance Plethysmography. Charles C Thomas, Springfield, Illinois, USA

23 Kubicek, W.G., Kottke, F.J., Ramos, M.U. et al. (1974) The Minnesota Impedance Cardiograph - theory and applications Biomed. Eng.

24 " J.M. Porter, I.D. Swain, (1987), Measurement of cardiac output by electrical impedance plethysmography

25 Andrei Krivoshei, Mart Min, Toomas Parve, Ants Ronk, (2006), An Adaptive Filtering System for Separation of Cardiac and Respiratory Components of Bioimpedance Signal, Tallinn University of Technology

26 surf2stl [www]
http://www.mathworks.com/MATLABcentral/fileexchange/4512-surf2stl (07/06/2015)

27 stlwrite [www]
www.mathworks.com/matlabcentral/fileexchange/20922-stlwrite-write-binary-or-ascii-stl-file/content/stlwrite.m (07/06/2015)

Lihtlitsents lõputöö üldsusele kättesaadavaks tegemiseks ja reprodutseerimiseks

Mina Irina Konovalova (a*utori nimi*) (sünnikuupäev: 4.11.1989)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Dynamic thorax modelling system for bioimpedance simulation based on 4D human phantom,

(*lõputöö pealkiri*)

mille juhendaja on Rauno Gordon,

(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas TTÜ raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas TTÜ raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

_____ (*allkiri*)

_____ (*kuupäev*)

Lisa 2

METAANDMED

Töö pealkiri (eesti keeles): 4D inimese kehal põhinev dünaamilise rindkere modelleerimise süsteem bioimpedantsi simulatsiooni jaoks

Töö pealkiri (inglise keeles): Dynamic thorax modelling system for bioimpedance simulation based on 4D human phantom

Autor: Irina Konovalova

Juhendaja(d): Rauno Gordon

Kaitsmise kuupäev: 11.06.2015

Töö keel: est / eng / rus: eng

Asutus (eesti keeles): TTÜ / TTÜ õppeasutus (nimi): Tallinna Tehnikaülikool

Asutus (inglise keeles): TTÜ / TTÜ õppeasutus (nimi): Tallinn University of Technology

Teaduskond (eesti keeles): Infotehnoloogia teaduskond

Teaduskond (inglise keeles): Faculty of Information Technology

Instituut (eesti keeles): Thomas Johann Seebecki elektroonikainstituut

Instituut (inglise keeles): Thomas Johann Seebeck Department of Electronics

Õppetool (eesti keeles): Siduselektroonika õppetool

Õppetool (inglise keeles): Chair of Communicative Electronics

Märksõnad /kui on/ (eesti keeles):

Märksõnad /kui on/ (inglise keeles):