Supporting Information

Documentation of the software-pipeline described in the paper "A flexible open-source pipeline for simulating transcranial electric stimulation"

Benjamin Kalloch
Pierre-Louis Bazin
Arno Villringer
Mario Hlawitschka

Leipzig University of Applied Sciences
&
Max Planck Institute
of
Human Cognitive and Brain Sciences

December 11, 2019

Contents

1 Introduction

In this document, we describe the necessary steps to reproduce the pipeline presented in the paper "A flexible workflow for simulating transcranial electric stimulation in healthy and lesioned brains". The purpose of this software pipeline is the simulation of transcranial electric current stimulation based on computational models derived from individual MR-images. The workflow covers all essential steps from head model creation from MR data, electrode modeling, and positioning, simulation as well as visualization.

The segmentation workflow and the post-processing of the segmentation has been published and described in detail before [KBK+18].

For a detailed description of the remaining processing steps along with additional background information please refer to the corresponding paper.

2 Setup

The workflow necessitates two tools for the basic functionality: 1) The segmentation and post-processing of the segmentation images are realized in the Medical Image Processing And Visualization tool (MIPAV) [MLM+01] with the Java Imaging and Science Toolkit (JIST) [LBC+10] and the CBSTools. Their setup will not be further discussed here as it has been described before[1][KBK+18]. 2) The OpenFOAM software package is used for setting up and simulating the test cases, i.e. the tdcs simulations. We will shortly describe where to obtain this package, which other software is required and how to set everything up.

2.1 System requirements

Hardware The workflow was tested on a workstation equipped with a 4-core Intel® Core™ i7 6700 processor clocked at 3,4 GHz with 32 GB of main memory and a 512 GB SSD drive. Major parts of the pipeline including the volume meshing as well as the solver application are not parallelized. For that reason, a processor with a high single-thread performance is preferred over a multi-processor system. In addition, both during segmentation as well as during the actual simulation intermediate results are written and re-read from the hard drive. Therefore, fast storage such as an SSD drive positively influences the processing speed. Storage requirements for a complete run of the pipeline with all intermediate results amount to approximately 30 GB which can be reduced if intermediate results are discarded after successful completion of the respective processing step.

Software The software pipeline is confirmed to work on Ubuntu 16.04 LTS and Ubuntu 18.04 LTS, but more recent versions of Ubuntu should work as well.

2.2 Meshing - CGAL, GMSH

We provide a custom application implementing the API of CGAL version 4.13.1. The version distributed with Ubuntu 19.04 is lacking an important fix. More recent

[1] https://figshare.com/s/f2e47b41e974ab67039b

versions of Ubuntu distribute a more recent version of CGAL, which may be incompatible with our meshing tool. For this reason, your must either compile CGAL 4.13.1 from the source or build a Docker container using the Docker file we provide (in the "docker" directory of *meshheadmodel* sub-module). This Docker file creates a Docker container which is capable of compiling and running the meshing tool. To build the Docker container execute

```
docker build -t cgal_and_gmsh_compiled_with_tbb_ubuntu1904 .
```

Before doing so, make sure to place the (unpacked) source code directories of GMSH 4.3 (from: `http://gmsh.info/src/gmsh-4.3.0-source.tgz`) and CGAL 4.13.1 (from: `https://github.com/CGAL/cgal/releases/download/releases%2FCGAL-4.13.1/CGAL-4.13.1.tar.xz`) into the "docker" directory and name them *gmsh* and *cgal* respectively. They are required to compile the meshing tool and will be copied into the container and built subsequently, so you may delete them after building the container.

You may use the *docker_compile.sh* script to recompile the meshing tool when you've made changes to the source code.

When compiled in the container, the tool also must be launched from inside the container. You can do this by calling the *docker_run.sh* script which invokes the *runMeshingTool.sh* script from inside the container. When using docker to run the tool consider that all paths within runMeshingtTool.sh (both input files and output files) must be specified according to the Docker environment (i.e. the shared folder) NOT according to the host machine. The best practice is to use the predefined input & output directories and to use paths relative to the MeshHeadModel executable.

Note that for both docker scripts you may have to adjust the variables $BASE_DIR and $CONTAINER_NAME if you use a path/name other then predefined in the file.

2.3 Electrode Modeling & Positioning - Blender

The modeling and positioning of the electrodes is implemented as a plugin for the 3D modeling software Blender. The plugin is confirmed to work up to version 2.79b which you can install using the apt package manager or download it from `https://www.blender.org/download/Blender2.79/blender-2.79b-linux-glibc219-x86_`

64.tar.bz2/.

2.4 Test case setup & Postprocessing - OpenFOAM

OpenFOAM version 6 provides the framework for the setup and execution of the simulation test cases. Since we extend the package with a custom solver application you will need the source code of OpenFOAM and compile it. You can download the source code of OpenFOAM 6 here: https://github.com/OpenFOAM/OpenFOAM-6. For the the post-processing functionality, especially ParaView version 5.4, please download the "Third Party" software package as well https://github.com/OpenFOAM/ThirdParty-6. Since we provide custom plugins for ParaView you need the source code of ParaView and compile it.

2.5 Processing of diffusion-weighted data

Our pipeline allows the use of diffusion-weighted (DWI) data for conductivity tensor estimation. To process DWI data, we use MRTrix3 which can be obtained from http://www.mrtrix.org/download/. For MRTrix to work properly you will also need the FMRIB Software Library (FSL) https://fsl.fmrib.ox.ac.uk/fsldownloads_registration and the Advanced Normalization Tools which can be downloaded here https://stnava.github.io/ANTs/.

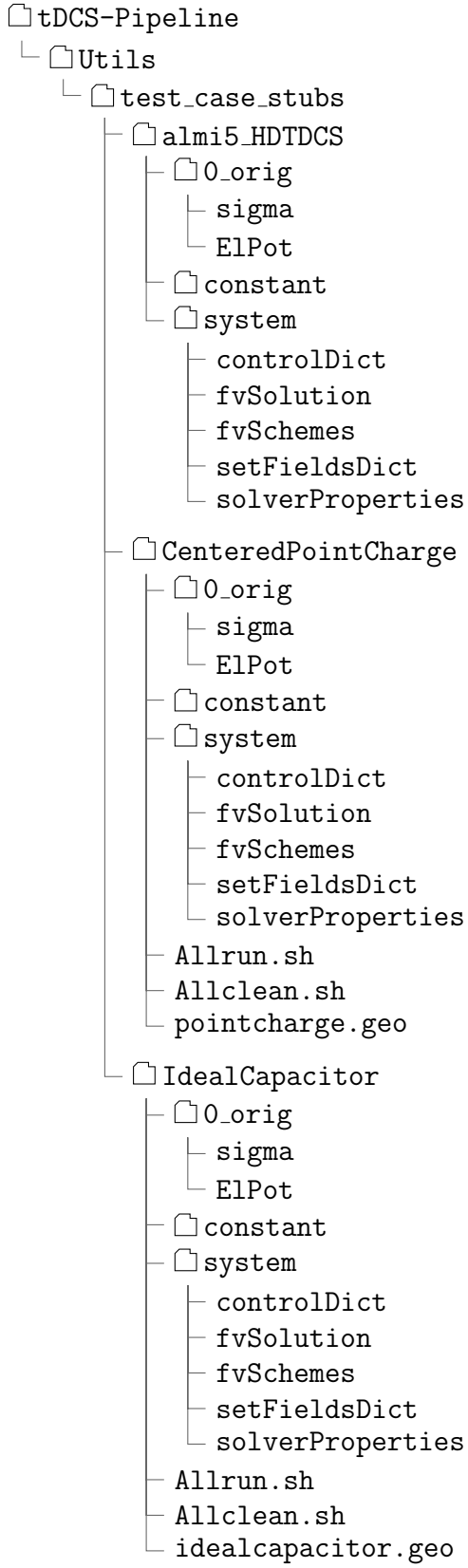
In addition, we provide plugins for ParaView v.5.4 that implement the computation of the conductivity tensors. The version of ParaView that is distributed with OpenFOAM is sufficient for that purpose. For some manual tasks functionality of MIPAV is required.

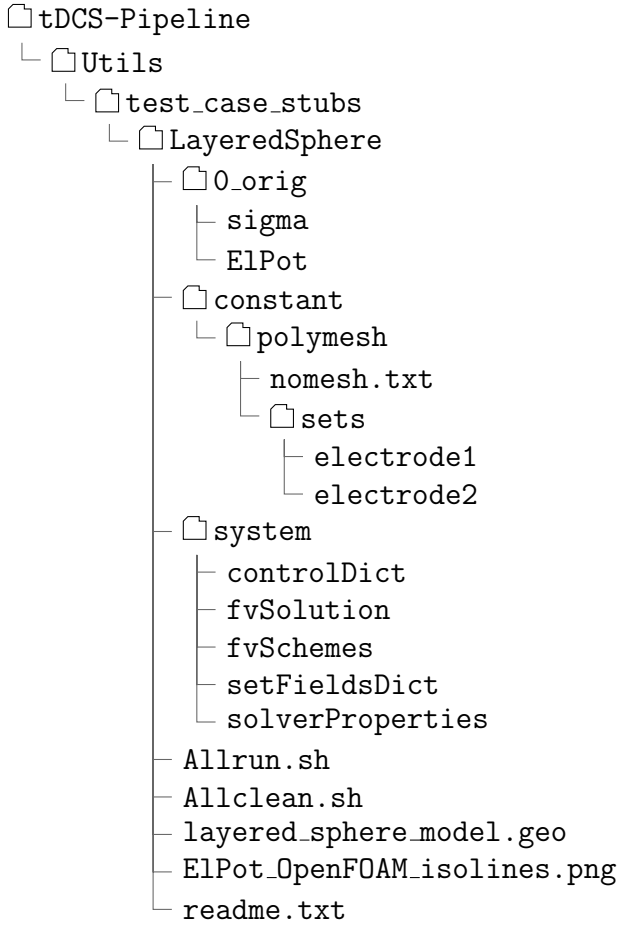
3 Structure of the Repository

This repository consists of several version controlled sub-modules. To update the submodules run "git submodule update". Please refer to the following tree list for an overview of the directory structure of the repository and its submodules. In the next sections, when discussing the individual stages of the pipeline we will refer to that files.

```
tDCS-Pipeline
├── TDCSSolver
│   ├── Make
│   ├── TDCSSolver.C
│   └── createFields.H
├── MountElectrode
│   └── mountElectrode.py
├── MeshHeadModel
│   ├── deploy
│   │   ├── input
│   │   ├── output
│   │   └── runMeshingTool.sh
│   ├── docker
│   │   ├── Dockerfile
│   │   ├── docker_compile.sh
│   │   └── docker_run.sh
│   ├── CMakeLists.txt
│   └── mesh_head_model.cpp
├── SurfaceRemesh
│   ├── deploy
│   │   ├── input
│   │   ├── output
│   │   └── runMeshingTool.sh
│   ├── docker
│   │   ├── Dockerfile
│   │   ├── docker_compile.sh
│   │   └── docker_run.sh
│   ├── CMakeLists.txt
│   └── remesh_surface.cpp
└── ThreeLayeredSphereModel
    └── tdcsSphereModel_RushDriscoll.py
```

- tDCS-Pipeline
  - PVDiffusionToConductivityTensor
    - CMakeLists.txt
    - plugin.cmake
    - pqDiffusionToConductivityTensorPanel.cxx
    - pqDiffusionToConductivityTensorPanel.h
    - vtkDiffusionToConductivityTensorFilter.cxx
    - vtkDiffusionToConductivityTensorFilter.h
    - vtkDiffusionToConductivityTensorFilter_SM.xml
  - PVSymMatrixToGenMatrx
    - CMakeLists.txt
    - plugin.cmake
    - pqSymToGenMatrixPanel.cxx
    - pqSymToGenMatrixPanel.h
    - vtkSymToGenMatrixFilter.cxx
    - vtkSymToGenMatrixFilter.h
    - vtkSymToGenMatrixFilter_SM.xml
  - PVSaveOFField
    - CMakeLists.txt
    - plugin.cmake
    - FieldTemplate.cxx
    - FieldTemplate.h
    - TemplateString.h
    - pqSaveOFField.cxx
    - pqSaveOFField.h
    - vtkSaveOFField.cxx
    - vtkSaveOFField.h
    - vtkSaveOFField_SM.xml
  - Utils
    - documents
      - images
      - output
        - Location of this document
    - scripts
      - mapToFOAMField.py
    - test_case_stubs
      - almi5_DUAL
        - 0_orig
          - sigma
          - ElPot
        - constant
        - system
          - controlDict
          - fvSolution
          - fvSchemes
          - setFieldsDict
          - solverProperties


tDCS-Pipeline
Utils
test_case_stubs
almi5_HDTDCS
0_orig
sigma
ElPot
constant
system
controlDict
fvSolution
fvSchemes
setFieldsDict
solverProperties
CenteredPointCharge
0_orig
sigma
ElPot
constant
system
controlDict
fvSolution
fvSchemes
setFieldsDict
solverProperties
Allrun.sh
Allclean.sh
pointcharge.geo
IdealCapacitor
0_orig
sigma
ElPot
constant
system
controlDict
fvSolution
fvSchemes
setFieldsDict
solverProperties
Allrun.sh
Allclean.sh
idealcapacitor.geo


tDCS-Pipeline
Utils
test_case_stubs
LayeredSphere
0_orig
sigma
ElPot
constant
polymesh
nomesh.txt
sets
electrode1
electrode2
system
controlDict
fvSolution
fvSchemes
setFieldsDict
solverProperties
Allrun.sh
Allclean.sh
layered_sphere_model.geo
ElPot_OpenFOAM_isolines.png
readme.txt

4 The workflow

The workflow is divided into five major parts:

1. Segmentation
2. Electrode modeling & positioning
3. Volume mesh creation
4. Simulation case setup and execution
5. visualization and optionally
6. *conductivity tensor estimation (optional).*

In this chapter, we describe the basic workflow for each of the mentioned stages of the pipeline. For every stage of the pipeline, we describe the expected input and which output will be generated in a tabular format. You can replace individual stages with your own workflow given that your workflow can handle the same input data and will create the same output data. The order in which we describe the individual stages is the recommended processing order.

4.1 Segmentation

Input	Individual head MR image, T1-weighted (NIfTI file).
Output	Labeled image with one distinct label per structure (ANALYZE file)
Tasks	In this step of the pipeline, an individual head MR images is segmented and based on this segmentation, a labeled image is created.

We have published an extended description of the segmentation workflow together with a user guide [2] before [KBK+18].
In the mentioned paper we describe the creation of surface-based head and torso models from individual MR images. For our tES pipeline, only a sub-component

[2] https://figshare.com/s/f2e47b41e974ab67039b

of the described workflow namely the segmentation of the head MR image is required. We perform the segmentation of the MR image of a subject's head in the same manner as described in that paper. We are able to segment skin, skull and enclosed air cavities, cerebrospinal-fluid, gray matter, and white matter. Likewise, the segmentation images are post-processed in the same manner. Finally, the result of this segmentation workflow that we use further in our simulation pipeline is a labeled image that contains the desired structures.

Note that other than described in the mentioned paper the topology of the segmented structure does not necessarily need to adhere a strictly nested arrangement. With our image-based meshing approach we can create the volume mesh of structures of arbitrary arrangement, i.e. structures may intersect each other or have common interfaces. This property is especially important when adding pathological structures such as brain lesions after a stroke.

4.2 Electrode Modeling

Input	Iso-surface of the skin segmentation (STL file).
Output	Surface description of the electrode, (OFF file)
Tasks	In this step of the pipeline, the electrodes modeled (shape & location on scalp). A surface description of the final electrodes is created.

The electrode modeling and positioning is implemented as a plugin for the 3D modeling software Blender. To install the plugin in blender copy & paste the file *mountElectrode.py* into the plugin folder of your Blender installation. In Ubuntu this folder is usually located in the ".config" directory of your home directory:
`$HOME/.config/blender/2.79/scripts/addons/` .
In Blender enable the plugin via: File ⇒ User Preferences ⇒ Search in the list of plugins for "Object: Mount Electrodes" ⇒ tick it.

After successful installation you will be presented with the following user interface.

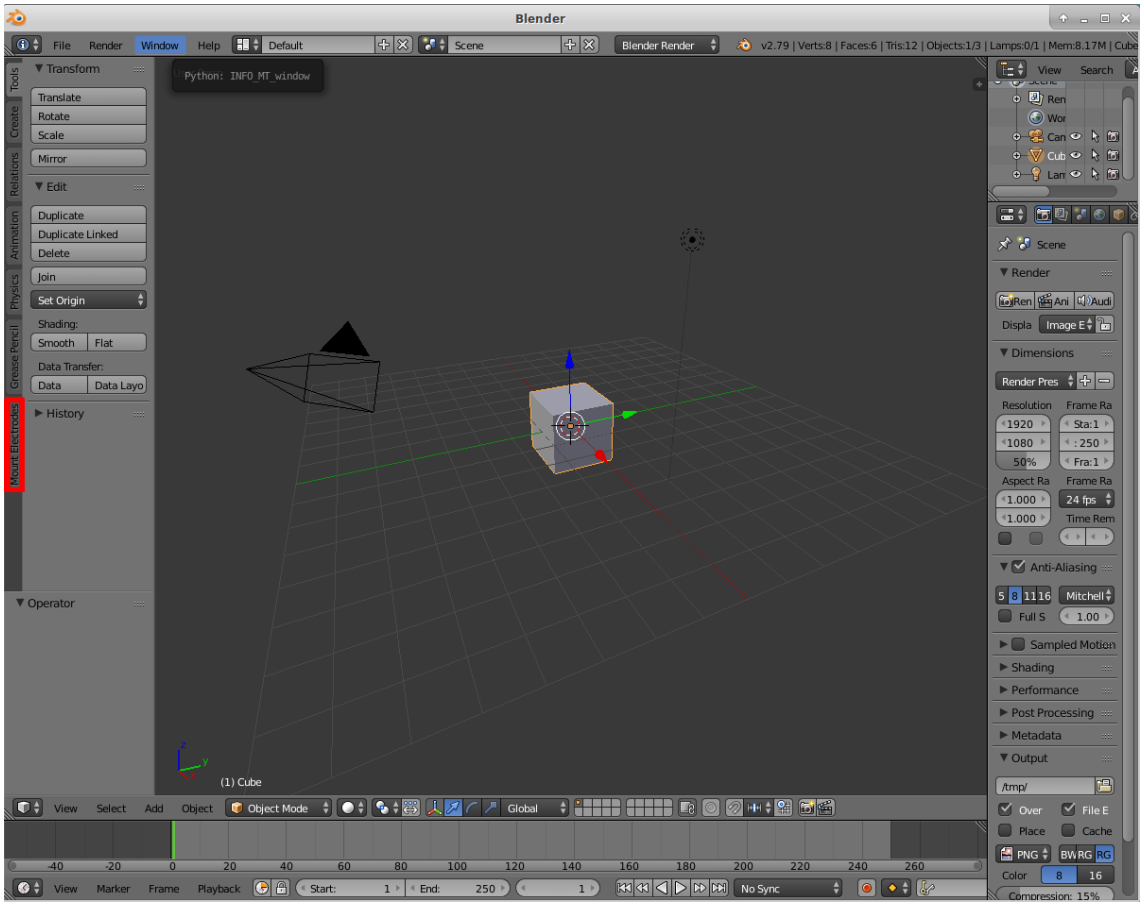


Figure 1: *Standard Blender UI after enabling the Mount Electrodes plugin.*

You can activate the UI of the plugin by clicking the “Mount Electrodes” tab on the left pane.

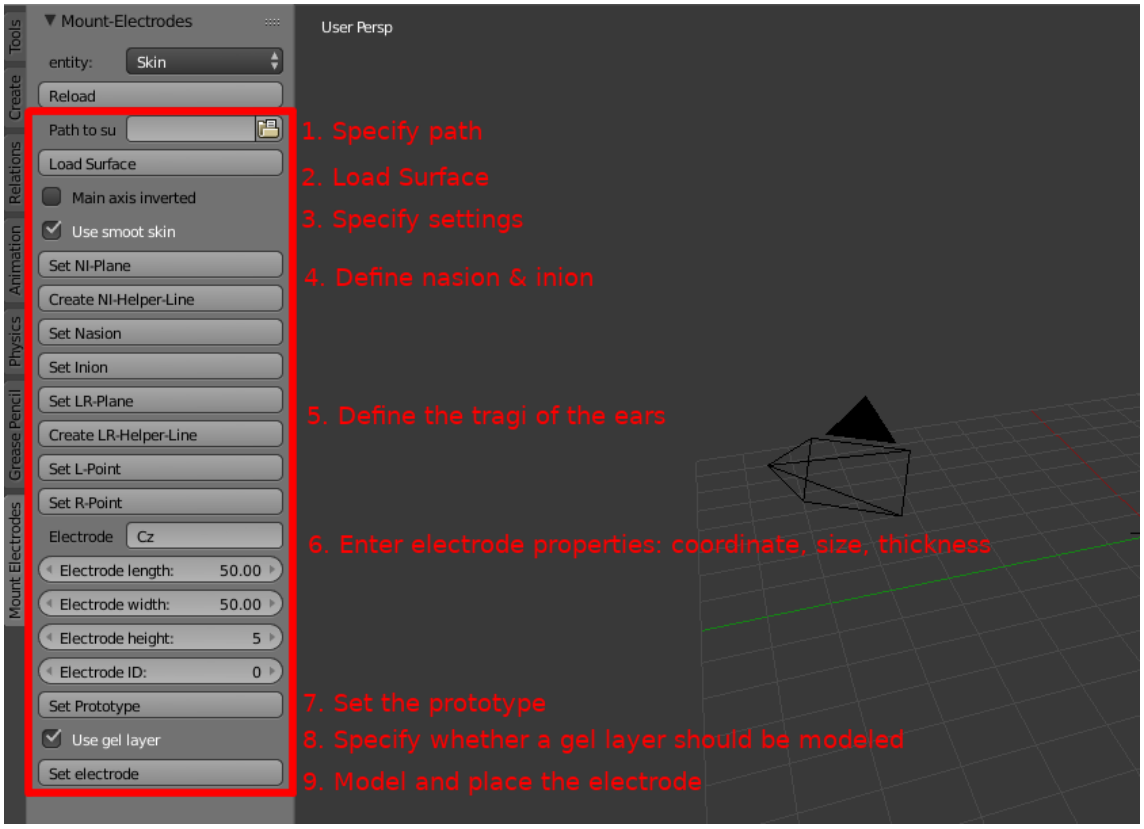


Figure 2: *UI of the Mount Electrodes plugin.*

First, you need to specify the path to an STL surface file that describes the surface of the skin of the model you want to mount the electrodes to. This surface must be generated in external software, for example, in ParaView using the "Contour filter" in ParaView from the binarized version of the labeled image file that used during volume meshing later. Then click 'Load Surface". This step may take a couple of minutes as the plugin computes a smoothed version of the input surface simultaneously.

The checkbox "main axis" inverted defines whether the head surface faces the down or up the z-axis. If your subject is oriented in the standard axial orientation, it is not required to check this.

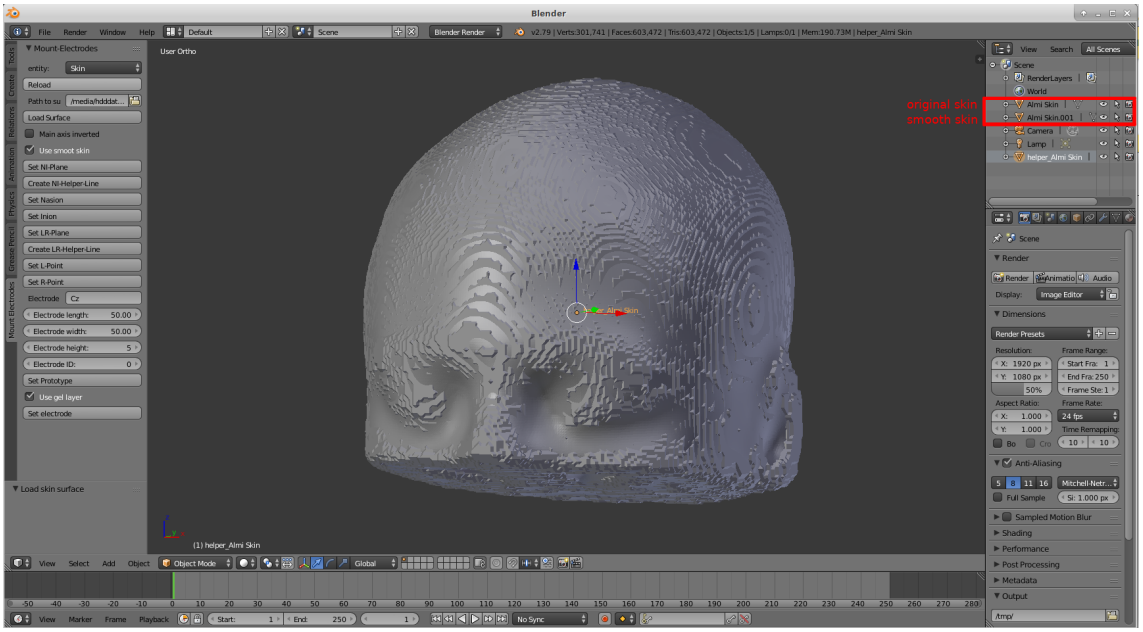


Figure 3: *Loaded skin surface. Notice how both, the original version as created by the Marching Cubes algorithm as well as a smooth version (mainly covered by the original version) are shown.*

The option “use smooth skin” defines whether the original skin surface or the smoothed version of the skin surface should be used for the electrode modeling. You should use the original surface if you are attempting a full image-based meshing of the head. Otherwise, to perform a surface-based meshing of the skin and an image-based meshing of all other tissues use the smoothed version to benefit from the greater smoothness and thereby more accurate modeling of the edges of the electrode. Note that the electrodes will always be meshed by a surface-based approach regardless of the skin will be meshed by an image-based or surface-based approach.

Your next task is to align the nasion-inion plane to the center of the head. The center position is precomputed by the plugin, but a correction might be necessary. You can manipulate the location using the red arrow in the center of the plane. If the head is tilted a rotation of the plane might be necessary. To perform a rotation, use the “R” button on the keyboard followed by the letter of the axis around which the plane should be rotated. Once you are satisfied with the alignment of the NI-plane click “Create NI-Helper-Line”. On the created line, click with the right mouse button at the location of the nasion, the chosen point will be highlighted at the

ni-line, then click “Set Nasion”. The plugin computes the location of the nasion automatically, but a correction might be necessary. Next, right-click at the location of the inion, then click “Set Inion”. You should notice that the length of the NI-line was reduced.

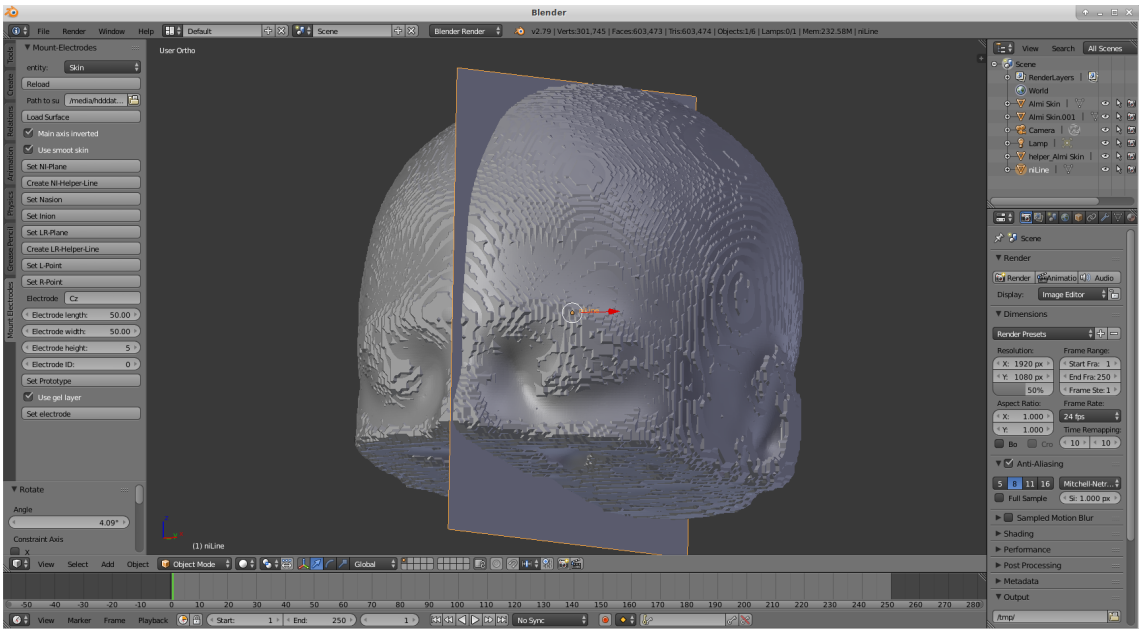


Figure 4: *NI-plane set.a*

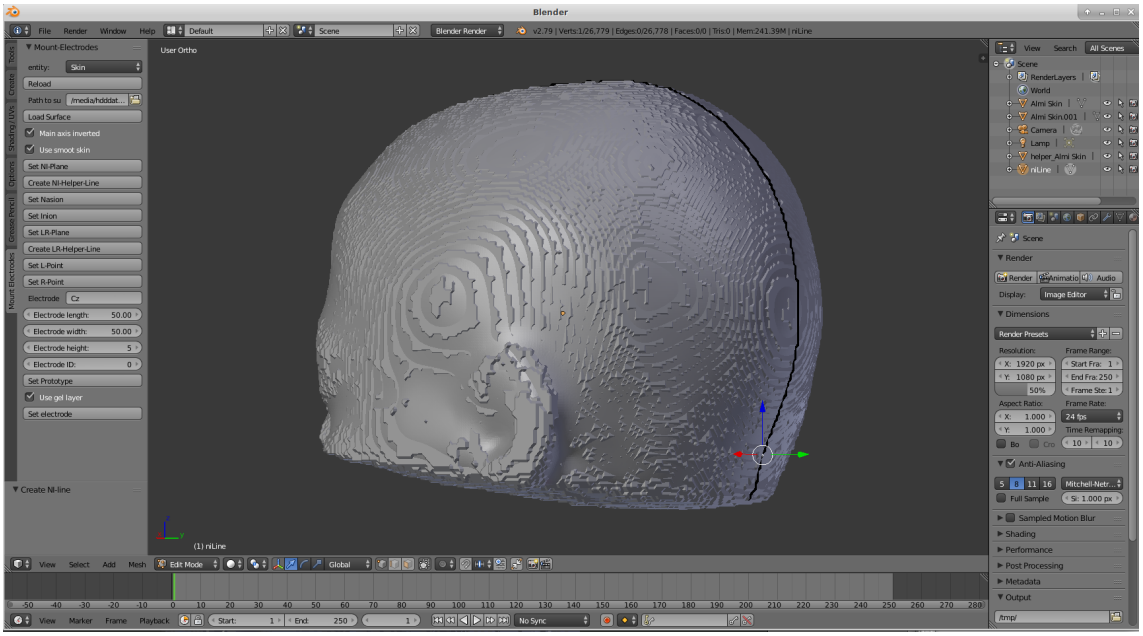


Figure 5: *Inion defined-plane set.*

Perform the definition of the tragi of the ears in a similar fashion: first, align the helper plane, create the lr-line and define the point on the right and on the left.

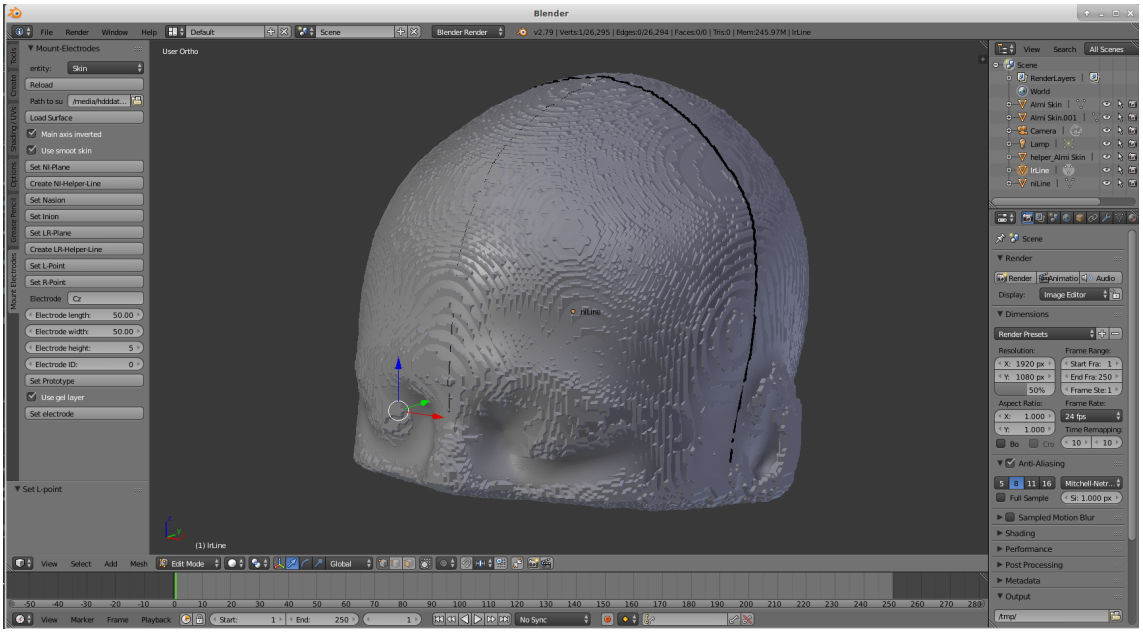


Figure 6: *LR-line defined.*

Next, the properties of the electrodes must be entered. First, enter the position according to the international 10-12 system. Then define the shape parameters of the electrodes. You do not need to modify the electrode-id as this will be auto-incremented with every new electrode you place.

After defining the electrode parameters click “Set Prototype”. A cube of the specified size will appear the specified location. You can adjust the position of the cube manually if the placement is unsatisfactory. You could even change the shape of the cube by modifying its geometry. This is how we modeled the triangular electrode.

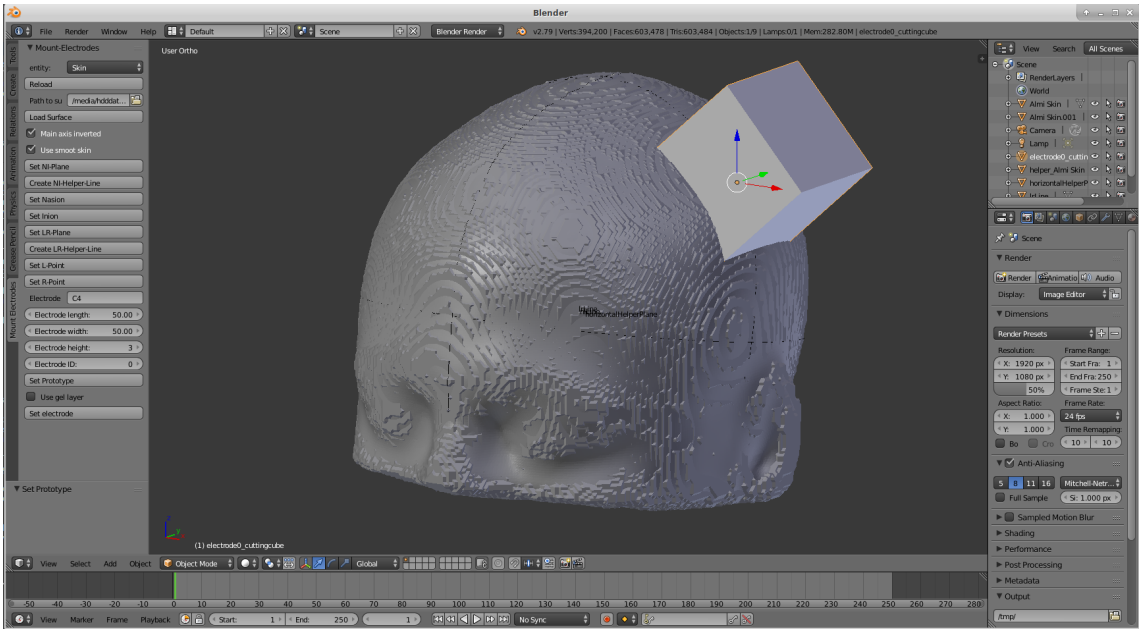


Figure 7: *Prototype set at C4.*

You may finally decide to use a gel layer underneath the electrode and the model and place the electrode by clicking "Set electrode". The electrode will be placed at the location of the cube, with the same extents and the specified thickness.

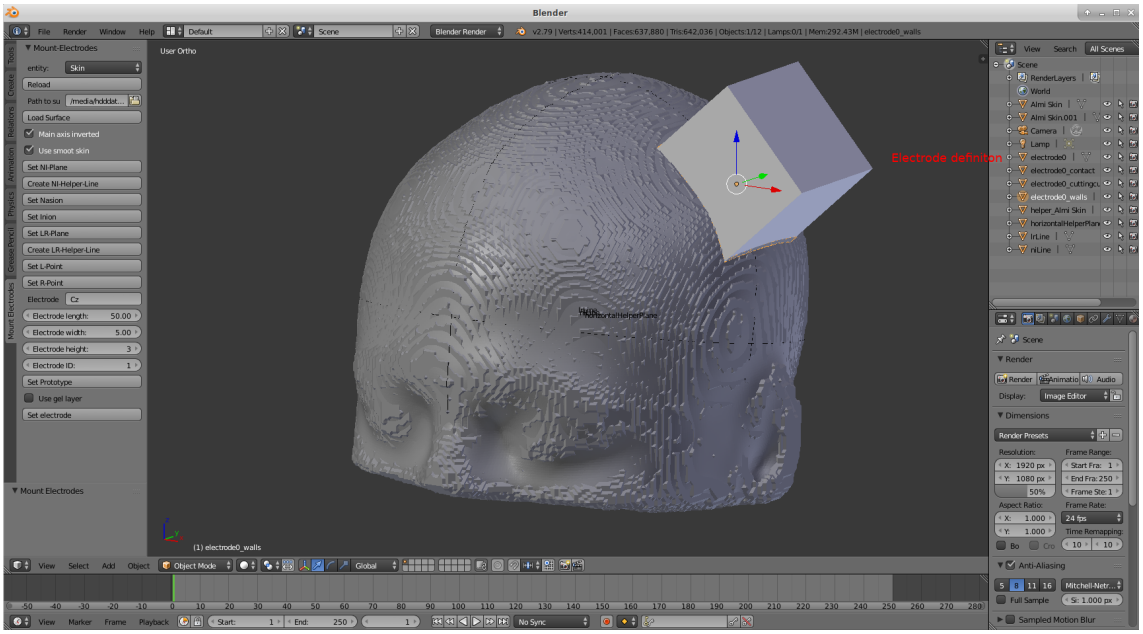


Figure 8: *Electrode modeled and ready for export.*

You can export the electrode by left clicking the name "electrodeX" (where X

= id of the electrode) and then *File* ⇒ *Export* ⇒ *STL*. Make sure to check the option "Selection Only" and then specify the file name and path where to save the electrode.

To use the surface description of the electrode in our meshing tool we must convert it into the Object File Format (OFF). We do this in Meshlab. Import the STL file of the electrodes. When importing Meshlab prompts to unify duplicate vertices. Do this. Next, reduce the number of triangles and clean the surface description. If you modeled the electrodes according to the original surface, i.e. the electrode doe not have a smooth surface, but this typical jagged surface due to the Marching Cubes algorithm then use the filter *Filter* ⇒ *Remeshing, Simplification, Reconstruction* ⇒ *Simplification: MC edge collapse*. Else, i.e. you modeled the electrode using the smooth skin surface and therefore also have a smooth electrode, use the filter *Filter* ⇒ *Remeshing, Simplification, Reconstruction* ⇒ *Quadaratic Edge Collapse Decimation* and specify the parameters as "Quality = 1", tick "Preserve Boundary of the mesh", and tick "Planar simplification". Finally "Export Mesh as" an OFF file under the File menu.

4.3 Volume Mesh Generation

Input	*Option 1)* Labeled image containing all segmented head structure (ANALYZE file) + electrode definitions (OFF file) *Option 2)* Labeled image containing all segmented head structures but the skin structure (ANALYZE file) + smooth surface representation of the skin (OFF file) + electrode definitions (OFF file) *Option 3)* Labeled image containing some irregular structures (e.g. lesioned tissue) (ANALYZE file) + surface representation of other tissue structures (e.g. skin, skull, CSF, GM, WM) (OFF file) + electrode definitions (OFF file)
Output	Tetrahedral volume mesh of the head with the electrodes mounted, boundaries of the electrodes represented as "Physical Surfaces" (GMSH file format v2).
Tasks	In this step of the pipeline a tetrahedral volume mesh is generated using a combination of an image-based as well as surface-based meshing approach.

The volume mesh generation is implemented as a C++ tool using the API of CGAL v. 4.13.1 and GMSH v. 4.3. Follow the instructions of the README file of the MeshHeadModel submodule to compile the application. For convenience, we provide a Docker file that creates a Docker container based on Ubuntu 19.04 that contains all required dependencies. You can then compile and execute the application within the container using the two provided scripts. All that is left to do in that case is to adapt the paths and container name in the scripts to your configuration and to download, extract and compile the GMSH source code.

The mode of standard operation is an image-based meshing meaning that in any case, you must at least provide a labeled image file in the ANALYZE file format as an input and the name of the output mesh. All other arguments are optional.

The image-based meshing operation will create a tetrahedral volume mesh with one separate mesh compartment (in GMSH terminology: 'Physical Volume') per label found in the input image. Since the boundaries of these compartments are

determined dynamically via bisection during the meshing process, the labels within the input images do not need to adhere any special constraints in terms of the topology of the represented structure. The individual mesh compartments are named in a numerically increasing order representing the order of their corresponding labels in the input image file. For compatibility with OpenFOAM, the boundary of the generated mesh compartments are not explicitly saved when exporting the mesh to the GMSH file format meaning that no separate boundary surface (in GMSH terminology: 'Physical Surface') of the structures in the input image will be generated. The reason for this is when converting the GMSH mesh into the OpenFOAM file format any preserved boundaries will be converted to an OpenFOAM 'patch' at which boundary conditions may be defined. However, by definition mesh boundaries or 'patches' must only be defined at the outer boundary of the mesh. A definition of an internal 'patch' is invalid and will raise a warning during conversion.

To include electrodes into the model you must specify the path to their surface descriptions in Object File Format (OFF) via the parameter 'electrodefiles'. This surface will be meshed using a feature-preserving surface-based meshing approach. Therefore, they must be of good quality as discussed at the end of section 4.2. Features edges with an inner angle smaller than 60 degrees will be preserved. The electrodes will be represented both as individual mesh compartments, i.e. 'Physical Volumes', and by their boundaries, i.e. 'Physical Surfaces'. Preserving an explicit description of their boundary is required to define the boundary conditions of the simulation later in OpenFOAM. The volume compartments of the mesh representing the electrodes will always be the last 'n' compartments (with n = #electrodes). Together with the input image the order of mesh compartments is then: image-based structures ⇒ electrodes.

The boundary of the meshed structures generated by the image-based meshing algorithm may not be as smooth as a smoothed surface-based meshed. This is especially disadvantageous in the case of the electrode representations and their contact surface with the scalp. Therefore, you may provide a smooth surface file of the skin and the corresponding smooth electrodes. You must specify the path to the smooth skin surface description with the parameter 'tissuesurfaces', again as an

OFF file. The quality of this surface file must be adequate. Note that in this case, the order of mesh compartments is as follows: image-based structures ⇒ smooth skin ⇒ electrodes. Again, the boundary surface of the skin compartment is not exported to the GMSH file format to avoid incompatibilities in OpenFOAM.

Finally, if your application requires a more precise approximation of the boundaries of the structures than it is achieved using the image-based meshing approach you may provide the surface definitions of additional internal structures (as OFF files) which will then be meshed using the surface-based meshing approach. Note that these surfaces must be of very good quality. To achieve adequate surfaces, we recommend the application of the *Taubin smoothing* functionality of MeshLab to first, smooth the surface, then the tool *MeshFix* by Marco Attene [Att10] to ensure an appropriate mesh quality and finally our *surface-remesh tool*. The order of the provided surfaces must begin with the most outer structure (i.e. usually this will be the skin surface) and it must end with the most inner structure (e.g. the gray matter structure). Additional inner structures may still be represented using a labeled image, for example, to represent lesioned tissue or the ventricles. A typical use case is to create the volume mesh based on surfaces for the typical structures (electrodes, skin, skull, cerebrospinal fluid, gray matter, white matter) and add irregular tissue such as lesioned tissue by providing a corresponding labeled image. It is completely unproblematic if the structures in the image intersect the provided surfaces in the final volume mesh. The intersection will also be visible in the final mesh. In the final volume mesh, the order of mesh compartments is as follows: image-based structures ⇒ skin ⇒ skull ⇒ csf ⇒ gm ⇒ wm ⇒ electrodes. Again, only the boundary surfaces of the electrodes is exported to the GMSH file format to avoid incompatibilities in OpenFOAM.

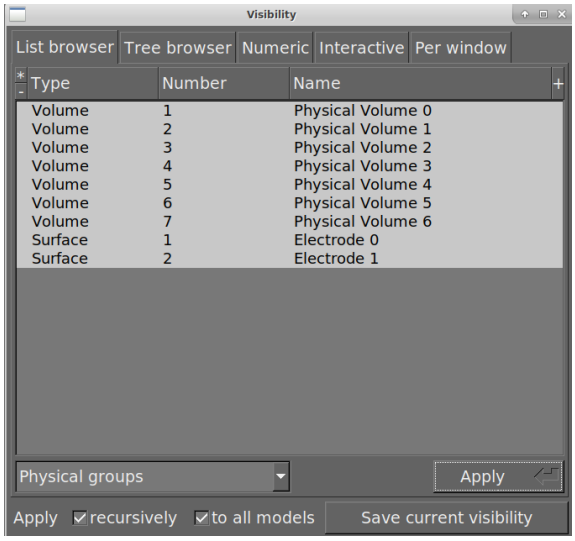


Figure 9: *Mesh compartments and boundaries of the almi5 test mesh using a smooth skin representation. Physical Volume 0 = skull, Physical Volume 1 = CSF, Physical Volume 2 = GM, Physical Volume 3 = WM, Physical Volume 4 = skin, Physical Volume 5 = volume of electrode 0, Physical Volume 6 = volume electrode 1, Electrode 0 = boundary definition of electrode 0, Electrode 1 = boundary definition of electrode 1.*

Additional parameters concern the mesh element size and quality (`https://doc.cgal.org/4.12/Mesh_3/index.html#title11`) and the mesh optimization phases (`https://doc.cgal.org/4.12/Mesh_3/index.html#title6`). Please refer to the provided online resources of the CGAL library for their explanation. The configuration of parameters that we typically use is predefined in the script file 'runMeshingTool.sh'.

4.4 Simulation Case Setup

Input	Tetrahedral volume mesh of the head. (GMSH file format v2)
Output	Electrical field strength (E), Electrical Current Density (J), Magnitude of E and J. (OpenFOAM file format)
Tasks	In this step of the pipeline we perform the electric field calculations.

We use OpenFOAM as a software suite to perform our simulations. Therefore,

you first must make sure to set OpenFOAM up properly and compile our TDCS-Solver accordingly.

OpenFOAM provides a set of command line tools to set up and execute a test case. Therefore a strict structure of the simulation directory must be maintained:

- **0** - The 0 directory contains the boundary conditions and initial values for all fields. Each field is represented by one file containing a so-called dictionary the defines the internal field values and the boundary conditions.

  The solver application will create more directories with numeric names representing the iteration in which the solution converged. The directory with the highest number contains the result.

- **constant** - The constant directory contains all static properties of the test case. In our case of tES simulation, this is only the mesh.

- **system** - The system directory contains all configuration files to control the setup and execution of the test case. These include: a) *controlDict* (information about the employed solver application, runtime behavior, maximum number of outer iterations, as well as precision, format and frequency of writing results), b) *fvSchemes* (numerical discretization schemes of the involved operators, in our case for the divergence-, gradient- and Laplace-operator), c) *fvSolution* (control of the equation solver, solver tolerance, maximum/minimum number of iterations, relaxation factors, number of non-orthogonal corrector iterations), d) *setFieldsDict* (initial field values for all mesh compartments, in our case the initial conductivity values), e) *solverProperties* (parameters for our TDCSSolver, number of electrodes, name of the contact surface of the electrodes (always 'faceZone_n' with n=[0...(#electrodes - 1)]) and the desired input current strength.

Please refer to the 'test_case_stubs' of the 'utils' submodule for an exemplary test case folder setup. Note that this basic structure can be copied to any new test case. To prepare a test case folder for a new test case the following steps must be executed:

1. **Import the mesh**: Using the tool *gmshToFoam* you can import the GMSH volume mesh files created by our meshing tool. After successful import, downscale the mesh accordingly since 1 unit of measurement corresponds to 1m in OpenFOAM. Therefore, a mesh generated from an MRI of 1 mm isotropic resolution must be downscaled by 0.001: *transformPoints -scale '(0.001 0.001 0.001)'*.

2. **Set boundary conditions**: Create the 0 directory with files for both the conductivity (*sigma*) and the electrical potential (*ElPot*). In bohth cases, the *boundaryField* dictionary must contain sub-dictionaries for each electrode (denoted by 'patchn' with n=[0...(1 - #electrodes)]). We apply a Dirichlet boundary condition in both cases. The 'defaultFaces' boundary must be present as well. This boundary represents all non-electrode outer boundaries of the mesh (i.e. the scalp that is not in contact with the electrodes).

3. **Set initial values**: We set the initial field values using the *setFields* tool. This tool reads the 'setFieldsDict' file. Therefore, you first must adapt that file according to your test case. You must specify the conductivity value for each mesh compartment in that file. Note that mesh compartments are defined by *cellZones* in OpenFOAM. The order of the cellZones is the same as the order of the 'Physical Volumes' in GMSH. This order can be verified in GMSH using the visibility menu.

   After you successfully adapted the *setFieldsDict* file just run *setFields*. The tool will populate the 'internalField' dictionary of *sigma* for each cell of the mesh with the specified conductivity value according to their membership to the mesh compartments.

4. **Define application solver parameters**: You must specify the target input current strength, as well as the number of electrodes and the names of their contact surfaces. When importing the GMSH mesh into OpenFOAM the contact surfaces will be stored as *faceZones*. Therefore, when importing the mesh from our meshing tool you must specify the names separated by a white space as 'faceZone_n', again with n=[0...(1 - #electrodes)]).

5. **Run the simulation**: Depending on the quality of the input mesh you may have to adjust the convergence criteria in the *fvSolution* file. You may decrease the relaxation factor for an under- relaxation of the system or further decrease the 'tolerance' parameter of the solver solving the E field. In severe cases of bad mesh quality, you may enable a limiter in the Laplace scheme to avoid overshoots in the subsequent gradient calculations (laplacian(sigma,ElPot) Gauss linear limited 0.5). Our meshing tool typically generates meshes of adequate quality. Therefore, it should not be necessary to use any of these means.
   You may run the simulation by executing the *TDCSSolver*.

6. **Visualize the result**: Run *ParaFoam* to visualize the result. Please refer to `https://www.openfoam.com/documentation/user-guide/paraview.php` for an explanation for the ParaView user interface when visualizing an OpenFOAM test case.

4.5 Incorporating Diffusion-Weighted Imaging Data

Input	DWI data of the subject, b-value, b-vector files.
Output	Conductivity tensors (OpenFOAM field file format)
Tasks	In this step we preprocess the DWI data of the subject, compute the diffusion tensors, convert them to conductivity tensors and export them in OpenFOAM field file format.

The conductivity values of a simulation case can be enriched by information derived from DWI data of the subject. In order to incorporate this information we suggest a workflow based on MRTtrix3 for data preprocessing and ParaView for the computation of the conductivity tensors.

The recommended processing pipeline in MRTrix3 is:

1. **dwidenoise** - Perform denoising of the DWI data.

2. **dwipreproc** - Motion & eddy current correction.

3. **dwi2mask** - Create a brain mask.

4. **dwibiascorrect** - Correct magnetic field inhomogeneities (only in the brain using the computed mask).

5. **dwi2tensor** - Compute diffusion tensors.

6. **tensor2mestric** - Compute the fractional anisotropy map.

7. **fslmaths** - Remove NaN values from both the DTI image as well the FA map using 'fslmaths IMAGE_FILE_NAME -nan OUTIMAGE_FILE_NAME"

The next step is to register the DTI data into the space of the T1-weighted ANALYZE image. It is important to use the same ANALYZE image of the subject that was used for the mesh generation as well. This way the registered tensor data will align well with the white matter compartment of the mesh. Using *FSL Flirt*, we first perform an affine registration of the FA map to the T1 image and save the computed transformation matrix. Since both images are of different sequences we use the 'mutualinfo" cost function for this registration. Second, we perform a non-linear registration of the linearly registered FA map to the T1 image using *FSL Fnirt*. Again we save the warpfield describing the transformation. We then apply both the linear registration matrix and the non-linear warpfield to the DWI data using *FSL vecreg* and thereby ensure the preservation of the tensor orientation.

The subsequent conversion from diffusion tensors to conductivity tensors will be performed in ParaView. However, the NIfTI reader module of ParaView may not load NIfTI files with multi-component values such as tensor data properly. Therefore, we convert the NIfTI file to a vtkImage using *ITKSnap*.

The generate VTK file can be opened in ParaView. In the simulation directory launch 'paraFoam" since we need the mesh of the simulation case as well. Then perform the following sequence of filters to the DTI data (filter marked with * are custom filters that are provided as submodules of this repository):

1. **Transform filter** - Scale down the image file to match the extents of the mesh. This typically involves a downscaling by 0.001.

2. **SynMatixToGenMatrix filter** [*] - We inflate the 6-element symmetric

tensor to a 9-element tensor as:

$$\begin{vmatrix} a & d & f \\ \cdot & b & e \\ \cdot & \cdot & b \end{vmatrix} = \begin{vmatrix} a & d & f \\ d & b & e \\ f & e & c \end{vmatrix} \tag{1}$$

3. **DiffusionTensorToConductivityTensor filter** [*] - We convert the diffusion tensors to conductivity tensors according to volume-constraint method [WAT+06].

4. **ResampleWithDataset filter** - We resample the diffusion tensors onto the 'internal mesh" of the simulation case. Choose 'Conductivity Tensors" as 'Input" and 'Head Mesh" as the 'Source" in the filter options.

5. **PointDataToCellData** - OpenFOAM reads and writes field values as cell data, i.e. one value that is valid within the entire cell. However, our image data are defined as point data, i.e. data that are defined on the vertices of the mesh. To successfully transfer the conductivity tensors to an OpenFOAM field we interpolate the data defined on the mesh points to the cells. Note that the filter must be executed on the resampled dataset.

6. **SaveAsOFField** [*] - The last task will be to export the field data into an OpenFOAM conform field data format. Use this custom filter and specify a target path for the output file. Alternatively, you may export it as a CSV file in a list format.

Finally, you may transfer the exported field data onto the *sigma* field of the simulation case using the *mapToFOAMField.py* Python script. Expected input parameters are a) a list file containing the values that should be mapped, and b) the path to the OpenFOAM field file. Note that a) must not be an OpenFOAM field. Therefore, if you exported the conductivity tensors using the SaveAsOFField filter you must remove the OpenFOAM header of the file and the closing bracket at the end of the file and just retain the list of field values (one tensor-value per line).

References

[Att10] Marco Attene. A lightweight approach to repairing digitized polygon meshes. *The visual computer*, 26(11):1393–1406, 2010.

[KBK+18] Benjamin Kalloch, Jens Bode, Mikhail Kozlov, André Pampel, Mario Hlawitschka, Bernhard Sehm, Arno Villringer, Harald E Möller, and Pierre-Louis Bazin. Semi-automated generation of individual computational models of the human head and torso from mr images. *Magnetic resonance in medicine*, 2018.

[LBC+10] Blake C Lucas, John A Bogovic, Aaron Carass, Pierre-Louis Bazin, Jerry L Prince, Dzung L Pham, and Bennett A Landman. The java image science toolkit (jist) for rapid prototyping and publishing of neuroimaging software. *Neuroinformatics*, 8(1):5–17, 2010.

[MLM+01] Matthew J McAuliffe, Francois M Lalonde, Delia McGarry, William Gandler, Karl Csaky, and Benes L Trus. Medical image processing, analysis and visualization in clinical research. In *Computer-Based Medical Systems, 2001. CBMS 2001. Proceedings. 14th IEEE Symposium on*, pages 381–386. IEEE, 2001.

[WAT+06] Carsten Hermann Wolters, Alfred Anwander, Xavier Tricoche, D Weinstein, Martin A Koch, and Robert S Macleod. Influence of tissue conductivity anisotropy on eeg/meg field and return current computation in a realistic head model: a simulation and visualization study using high-resolution finite element modeling. *NeuroImage*, 30(3):813–826, 2006.