Software-Project 2017

# Review Document

## Real-Time Mesh Utilities

Petros Simidyan, Blerta Hamzallari, Felix Griesau, Marco Klamke
Julius Lerm, Lars Debor, Simon Heinke, Sugandha Sachdeva

last change: July 2, 2017

# 1 Introduction

The review document grants insight into the progress of the project and covers the decisions the team made during development.
It serves as an extension to the design and developer documents, which describe the structure and code of the software in more detail.
The document is divided into three segments, each representing a phase of the project. Each phase stands for specific tasks and gives a time frame for when they have to be finished.
Thus, after every phase progress gets documented and reflected and a lookout for further work on the project is provided.

# Contents

# 2 First Phase: Planning

This segment of the review document is divided into two parts. The progress of the first phase of the project is described. It contains the results of planning as well as the outcomes of the design process. Results of planning include the development model, cost and risk estimation, the milestones and organization. The design process contains the outcomes of the first iteration, arrangements for the next iteration and a list of tools.

## 2.1 Results of Planning

A large part of the first phase of the project (i.e. scheduling and draft) is reflected on the functional specification document. The requirement analysis is registered, the objectives are declared, whereas the decisions and the product information is written down.

### 2.1.1 Software Development Model

This section contains information about the software model chosen, based on the requirements of the project. The principals of the group, client requirements and knowledge about the project play an important role in choosing the development model. Based on the latter, the development team decides its work flow.

**Agile Development Model: SCRUM** The group chose SCRUM because it is an iterative and incremental agile software development framework for managing product development. The duration of each sprint was set to two weeks. Each phase of the software development has two sprints.

Every sprint ends with a presentation by the relevant working group about the developments and progress during the sprint. The end of the respective phase of the project is marked by a working prototype and a presentation which includes a summary of the work done by the entire team.

**Projects specific adaptation to the model:** Every person in the team has multiple roles. All group members work on both the documents and the code.

#### 2.1.1.1 Software Development Specific Content

Since the group decided for an agile development model, the milestones need to be stated and agreed upon by the team. Milestones are the aim or the expected output of each development phase. They help the team to specify what features should be completed by which deadline.

### 2.1.2 Effort Estimate

The main purpose of the effort estimate section is the categorization of the different parts of the project regarding their complexity and effort criteria. (see figure 1)

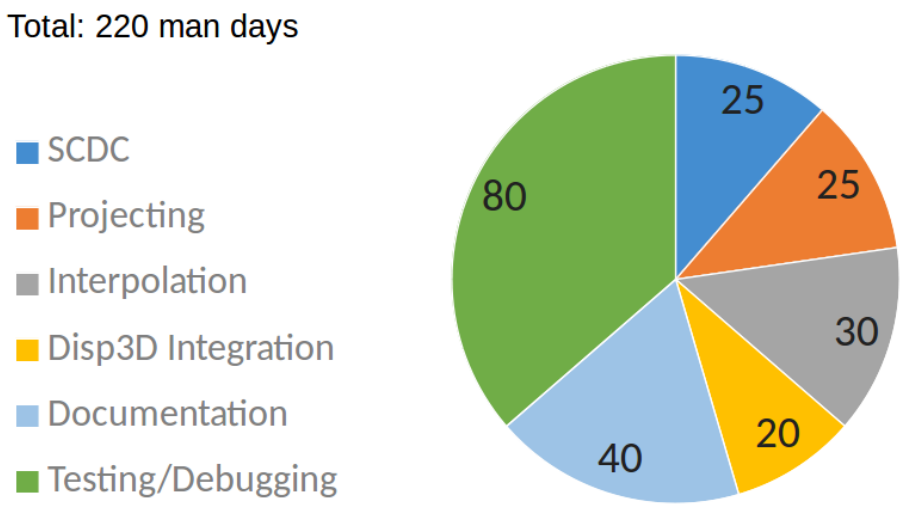


Figure 1: Effort estimate

### 2.1.3 Risk Estimate

In this section, the probabilities of different risks involved in the project are listed. This will help the team to determine what aspects of the implementation should get a higher priority.

**RE1:** Communication problems in the team

**RE2:** Coverage is too extensive

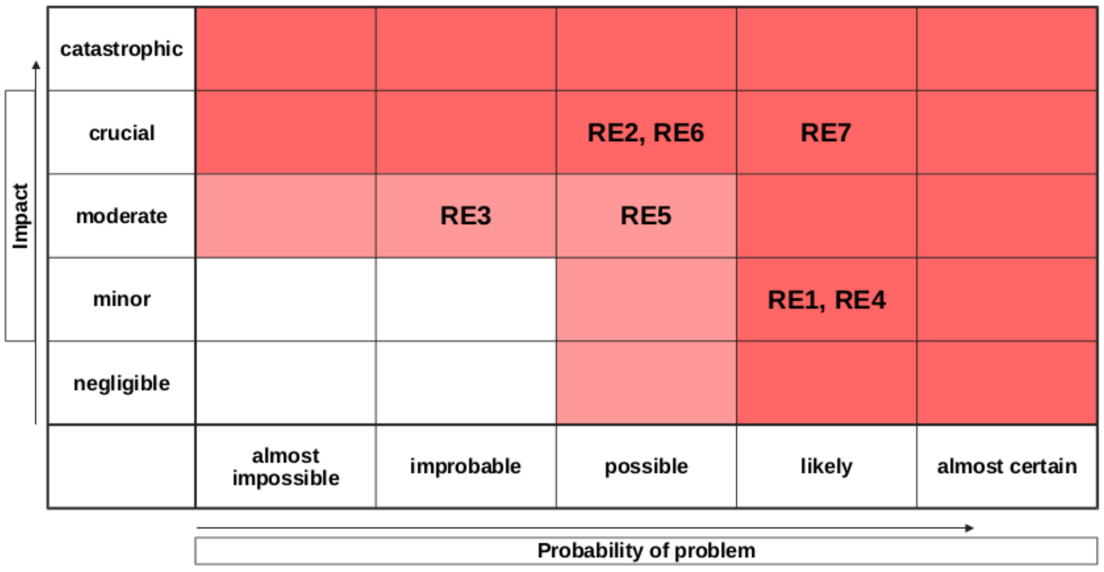
**RE3:** Framework does not provide the needed functionality

**RE4:** Resource bottleneck deriving from team members temporary absence

**RE5:** Change of requirements due to miscommunication with the product owner

**RE6:** Hidden complexity

**RE7:** Acceptable computation time takes a lot more effort than expected



| | | | | | |
|---|---|---|---|---|---|
| catastrophic | | | | | |
| crucial | | | RE2, RE6 | RE7 | |
| moderate | | RE3 | RE5 | | |
| minor | | | | RE1, RE4 | |
| negligible | | | | | |
| | almost impossible | improbable | possible | likely | almost certain |

Impact
Probability of problem

Figure 2: Risk estimate

#### 2.1.3.1 Handling Risks

In case one of the mentioned risks occurs, certain strategies for solving the issue are provided. Only the ones having crucial impact on the project are dealt with in the following section.

**RE2:** Optional requirements will only be addressed after every mandatory criteria is fulfilled.

**RE6:** The team communicates with the product owners to eradicate all ambiguities.

**RE7:** To keep the computation time as low as possible, changes to the software will be tested constantly to determine even minor impacts on efficiency.

### 2.1.4 Milestones

These Milestones provide major dates for significant events during the time of the project. Furthermore they serve as a guideline for the progress that is desired at a certain point of time.

| | | |
|---|---|---|
| **First Phase** | 03.05.2017 | First Phase Finished |
| **Second Phase** | 04.06.2017 | Detailed Design and Features Implemented |
| | 08.06.2017 | Second Phase Finished |
| **Third Phase** | 21.06.2017 | Features Tested and Optimized |
| | 02.07.2017 | Portation to MNE Scan |
| | 05.07.2017 | End of Project |

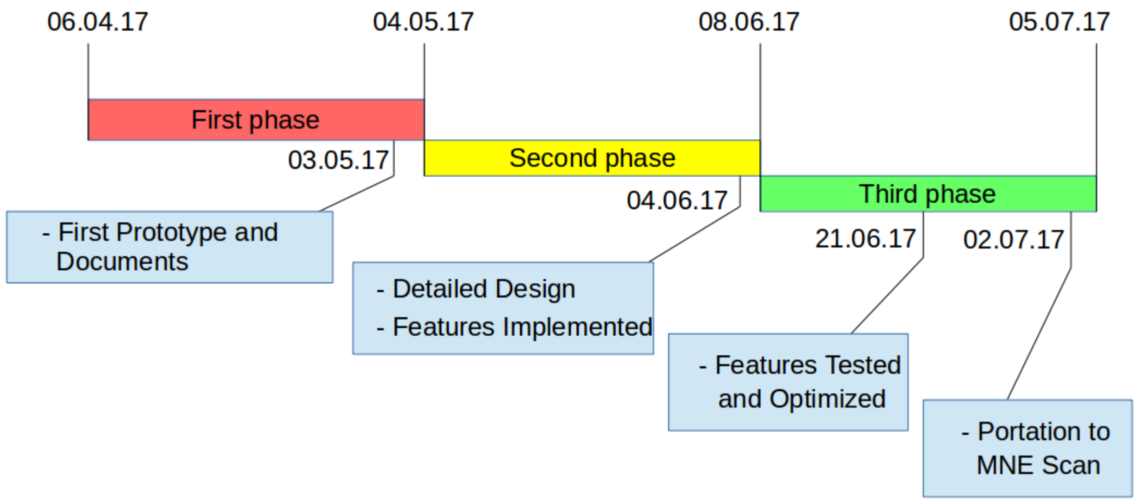


Figure 3: Timeline of milestones

#### 2.1.4.1 Checklist

**03.05.2017** First Phase Finished

1. Functional Specification
2. Preliminary Design
3. Review Document
4. First Prototype
5. Presentation

**04.06.2017** Detailed Design and Features Implemented

1. Detailed Design
2. SCDC (operational)

3. Projecting Algorithm (operational)
4. Interpolation Algorithm (operational)

**08.06.2017** Second Phase Finished

1. Presentation
2. Review Document
3. Integration into Disp3D

**21.06.2017** Features Tested and Optimized

1. SCDC (tested)
2. Projecting Algorithm (tested)

**02.07.2017** Portation to MNE Scan

**05.07.2017** End of Project

1. Review Document
2. Presentation
3. Interpolation Optimized

### 2.1.5 Organization

This section covers the rules, agreements and the partitioning regarding the teamwork in the project, so the work itself will be efficient and organized

#### 2.1.5.1 Ways of Communication

**Telegram:** Used for quick and direct team communication so that possible misunderstandings will be solved in no time.

**E-mail distribution list:** Used for scheduling the team meetings and communications with the extended team, including the product owners.

**Team meetings:** Used for the review and direct discussion of the encountered problems.

**Skype:** Used in the cases of the absence of a team member.

**Jira:** Used for scheduling tasks and keeping track of the progress done by each member of the team.

**Dropbox:** Used for exchanging documents and file sharing.

#### 2.1.5.2 Additional Agreements

- Internal team meetings (without product owners): Tuesdays and Thursdays at 19:00
- External team meeting (with the product owners): Wednesdays at 17:00
- Meeting of subgroups : upon consultation and demand

#### 2.1.5.3 Role Assignment in SCRUM

**Product Owner:** Thomas Jochmann, Lorenz Esch

**Scrum Master:** Simon Heinke

**Development team:** Blerta Hamzallari, Felix Griesau, Julius Lerm, Lars Debor, Marco Klamke, Simon Heinke, Sugandha Sachdeva, Petros Simidyan

**Client, User:** Participants of the MNE CPP project of Boston Children's Hospital

#### 2.1.5.4 Role Assignment Organization

**Advisor:** Thomas Jochmann, Lorenz Esch

**Team leader:** Simon Heinke

**Build master:** Lars Debor

**Version management:** Felix Griesau

## 2.2 Result of Design

The results of design can be found in the preliminary design document. The connections between different packages, components and classes are explained and visualized using UML-diagrams.

### 2.2.1 Tools

The used tools are programs which facilitate the different aspects of the organization and development.

#### 2.2.1.1 Organization Tools

**Code versioning:** The source code will be managed via *Git* to ensure a convenient exchange and tracking of code.

**LaTeX:** *LaTeX* is used for editing review documents and documentation.

**Doxygen:** This program will automatically document source code comments.

**Visual Paradigm:** Used for creating UML-diagrams and other aspects of visual documentation.

#### 2.2.1.2 Development Tools

**Development environment:** Since it is already established within MNE-CPP, *QtCreator* will be used for code editing and compiling.

**Program language:** *C++* will be used hence it is the main language of MNE-CPP

**Operating systems:** The software will be developed on Linux and Microsoft Windows.

**Libraries:** Besides standard libraries from the C++ 11 standard, the MNE-CPP framework is based on Eigen, OpenGL and Qt.

### 2.2.2 Results of first iteration design:

The results of the first iteration correlate with the respective milestone.

**Functional specification:** The requirements specification is transferred into the corresponding functional specification.

**Preliminary design:** The preliminary design includes a first overview of the main features and structure (see preliminary design document).

**Implementation:** A first executable prototype is created.

**Planning:** Milestones for the next iteration are declared.

### 2.2.3 Tasks for next iteration

**Refining the preliminary design:** The preliminary design is extended with diagrams and more detailed descriptions. This will be done according to milestones progress.

**Further implementation:** The mentioned features will be implemented and tested.

# 3 Second Phase: Implementation

This segment of the document contains the outcome of the second phase of the project. Basic concepts and decisions of the final design are mentioned and explained. Further the final design and a corresponding developer documentation are available as separate documents.

Alongside the achievements, problems that arose during this iteration are covered and some issues have to be revisited throughout the third phase.
Lastly a prospect for the next phase is given, with all the tasks that have to be taken care until the end of the project.

## 3.1 Bugreview

Due to encountering some problems during implementation, not all features are finished and ready for testing. Nevertheless, we were able to fix the majority of the occurring issues.
The following aspects and features need further processing in phase three.

### 3.1.1 Missing Feature

Until the end of the project this feature has to be implemented and tested.

**Integration into Disp3D:** After successfully implementing all algorithmic features into the MNE-CPP library, a further integration is needed to make the newly created functions usable. Disp3D provides the GUI for visualizing the interpolated data, while enabling the user to change settings like the color table.

A new *sensorDataTreeItem* is needed to integrate functions from the library. Due to high complexity more time is needed to finish implementation.

To fix this problem the task will have the highest priority at the beginning of the third phase, to quickly finish its implementation and start with further testing and optimization.

### 3.1.2 Missing Mandatory Function

This function is mandatory and therefore has to be completed during the third phase.

**Bad Channels:** Sometimes sensors do not deliver correct data and are marked as bad channels. These sensors should not have any impact on the weight matrix and the following interpolation.

Currently this information is not utilized during computation. Therefore misleading data is used and the results are not completely correct.

Bad channels are to be detected and considered during calculations.

### 3.1.3 Missing Optional Functions

After finishing the implementation of all mandatory functions these extensions are to be considered.

**Computation on GPU:** To further increase the efficiency of the interpolation, calculations are done using compute shaders, utilizing the GPU.

**Portation to MNE Scan:** All new features are accessible through the front-end application MNE Scan.

## 3.2 Results of Second Phase

In this section the achievements of the second phase are documented.

**Detailed Design:** A document for the detailed design of the software was created. It covers implemented classes and their structures, while giving explanations for all functions and components.
Further design choices made during development are described.

**Developer Documentation:** A developer documentation was created by using Doxygen. It is available as a separate document.

**Implementation:** Implementations of the first phase were continued and refined. The first three major features are operational and optimized to a certain degree.
This results in computation times within the limits that were agreed upon with the product owners.
The software is able to project sensors onto a surface, calculate the distances of vertices on a mesh and create the weight matrix for later interpolation, which is also implemented and runnable.

**Planning:** All remaining tasks are known and next steps for the coming iteration were discussed within the team.

## 3.3 Goals for Third Phase

In this section the goals for the following phase are outlined. All features mentioned in the Functional Specification are intended to be operational and optimized as the project ends after this phase.

**Testing of SCDC:** The correctness of the SCDC-feature has to be tested in order to guarantee realistic results when using its outputs during the later interpolation of signals. This could be done by manually verifying smaller sets of input data or by creating an alternative implementation that compares all possible paths and finds the shortest one.

**Testing of Projecting:** In case that the implementation of the sensor-to-mesh mapping will feature advanced data structures, the results can be verified by comparing them to the outputs produced by linear search.

**Portation to MNE Scan:** The features should ultimately be integrated into the front-end application MNE Scan, which requires for some extensions and adaption of the class interfaces. In the third phase we aim to achieve this concomitantly with the completion of the features.

**Optimization of Interpolation:** Interpolating the sensor data already provides a satisfactory level of efficiency at this point of time. In the upcoming phase we try to further improve its performance by utilizing compute shaders on the graphics card. This could speed up the calculation as well as offering the advantage of having the output data already on the graphics card for further rendering of graphical images.

**Bad Channels:** As the hardware sensors used to measure the brain activity are known to sometimes fail and thus to produce corrupt input data, these so called Bad Channels must be filtered out before any sensor signals are interpolated.

**Fibonacci Heap:** Since the method for the SCDC uses Dijkstra's algorithm, a so called Fibonacci Heap could speed up the calculation of distance tables and in this way allow for greater subsets of vertices (see Detailed Design for more details).

**K-D Tree:** An implementation of a balanced k-D tree could speed up the projecting algorithm, as it provides logarithmic run-time for lookups.

# 4 Third Phase: Validation

This segment of the document describes the results of the third and final phase of the project.
Unimplemented functions and bugs from the second phase are reviewed and solutions explained. In addition test cases and their results are described, to validate the correctness of the implementation.
At the end the whole project and its success is critically evaluated.

## 4.1 Testing

In order to validate the implemented algorithms, following tests were written.

### 4.1.1 Bad Channel Filtering

To eliminate the influence of bad channels (3.1.2), they are assumed to have an infinite distance to every other vertex. The test checks if all entries of columns that were identified as bad channels were set to infinity.
This success of this test showed that the function *GeometryInfo::filterBadChannels* indeed overwrites all relevant entries.

### 4.1.2 Empty Inputs

Because sturdiness is an important criterion for the program, empty inputs must not lead to a crash.

**Sensor Projecting** If the function for sensor projecting gets passed an empty vector of sensor coordinates it should return an empty vector of vertex IDs.
The success of this test showed that the function *GeometryInfo::projectSensors* indeed does not crash upon empty inputs.

**SCDC** If the function for surface constrained distance calculations gets passed an empty subset of vertices, it should calculate the full distance table for the passed mesh. This is verified by comparing dimensions of the output matrix.
The success of this test showed that the function *GeometryInfo::scdc* indeed returns the full distance table.

**Weight Matrix Creation** If the function for creating a weight matrix is passed an empty vector of sensor indices, it should return an empty matrix. This is verified by testing the size of the matrix object.
If the function is passed an empty distance table, it should give out a warning and return a null pointer. This is verified by testing if the output really is a null pointer.
The success of this test showed that the function *GeometryInfo::createInterpolationMat* indeed does not crash upon empty inputs and returns empty outputs.

### 4.1.3 Matrix Dimensions

Because the later live interpolation is achieved by multiplying a vector of sensor signals with the precalculated interpolation matrix, matching dimensions are crucial.
The output matrix of the SCDC must have as many rows as the number of vertices inside the used mesh and as many columns as the number of projected sensors.
The output matrix of the weight matrix creation must have the same dimensions as the passed distance table.
The result of a multiplication of sensor signals and the interpolation matrix must be a column vector that contains as many values as the number of rows inside the interpolation matrix.
The success of these tests showed that the program only produces matching dimensions and thus does not create arithmetic exceptions.

### 4.1.4 Coefficients of Interpolation Matrix

Since an interpolated vector of signals must not amplify the signal itself, the coefficients of a row inside the interpolation matrix must add up to one. This is verified by adding all entries of each row and checking if the result is one.
The success of this test showed that the calculated interpolation matrix indeed does not amplify or weaken the interpolated signal.

## 4.2 Bugreview

At the end of the second phase not all features were implemented and certain criteria was unfulfilled.(3.1)
This section provides a description of used solutions to handle the mentioned issues during the third phase.

### 4.2.1 Integration into Disp3D

The implementation of the *SensorDataTreeItem* took more time than expected, due to hidden complexity.
Because this feature represented the integration into the existing project MNE-CPP, understanding the given structure and code was necessary.
As it was the last unfinished feature of the project, more resources were shifted towards solving the issue and completing the task. Hence the amount of persons actively working on the feature was increased from originally three, to five team members.
Moreover these team members collaborated with the product owners to fulfill the task.
The mentioned actions resulted in a successful implementation.

### 4.2.2 Bad Channel Filtering

The problem was to find an appropriate method of when and how to filter bad channels during calculation.
Setting the bad channels to zero within the weight matrix of the interpolation would have resulted in assuming to not have any activity at the sensor location and thus weakening the activity of the area around it after the interpolation. This would not have been biologically correct.

The solution was to do the filtering directly after the distance calculation of the *SCDC*.
It is assumed that the flawed sensors are far away away from all other vertices and are given an infinite distance within the resulting distance table.
Therefore they automatically receive the value zero inside the weight matrix, while not damaging the resulting row sum of 1, which is important for testing purposes.

## 4.3 Critical Evaluation

As the project ends with the end of the third phase, a final overview serves as conclusion of this document.
This part critically evaluates the success of the project.

### 4.3.1 Evaluation of Project Success

Although all functional requirements were fulfilled, there are some restrictions concerning scale and efficiency.

#### 4.3.1.1 Constrained Distance Calculation

Because a surface constrained distance calculation would result in unreasonably large outputs for some surfaces, the input is reduced in most cases (see detailed design for more information). Since the interface still allows a full distance calculation and the latter hardly ever is needed, this restriction does not influence the fulfillment of the respective functional requirements.
Certain configurations of the distance calculation lead to extensive run-time, e.g. the usage of a high-resolution surface (i.e. a surface with immense amount of vertices). Since this is a rather common tradeoff between precision and low computation time and because the surface constrained distances must only be calculated once, this restriction does not interfere with the functional requirements.

#### 4.3.1.2 Interpolation Matrix

Depending on the passed parameters, the live interpolation of sensor signals takes up a significantly high run-time. When the amount of relevant sensors nodes is increased (see detailed design for more information), the interpolation matrix gets denser and thus produces higher computation time. Again, this is a compromise between precision and run-time and thus does not interfere with the respective functional requirement.

### 4.3.2 Evaluation of Project Organization

#### 4.3.2.1 Internal Communication

Due to weekly meetings and intensive communication via messengers, the internal communication of the team was overall good (formulierung verbessern).
To keep responsibilities clear, each task or subtask was assigned to a team member or a group of members. These members then were answerable for said task and were supposed to update the team about process made. While tasks were fulfilled correctly and in most cases promptly, status updates were sparsely made.

#### 4.3.2.2 Communication with Product Owners

With MNE-CPP being a rather large framework, communication with the product owners was crucial. While differences concerning the requirements or necessary changes of interfaces were mostly resolved swiftly, details about the frameworks architecture were sometimes not communicated. (vllt noch mehr schreiben)

#### 4.3.2.3 Tools and Software

The version management (i.e. Git) worked well throughout the course of the project. Because of inattentive use of the program, some unnecessary operation such as branch merging and reverting was necessary.
The issue tracking was used rather ineffective. Because some members misunderstood the concepts of issues, unnecessary or irrelevant issues were created and deleted without use. Functionality such as work logging and status updates was not used for the first phase of the project. Status updates of issues were only sparsely used for the rest of the project.

#### 4.3.2.4 Milestones and Sprints

The division of the project into several sprints was rather useless since left over issues had to be finished during the next sprint. This might be a result of the sprints being too short (two weeks) or wrong zusammenstellung der aufgaben.

#### 4.3.2.5 Effort Estimate

While the estimated working time was roughly correct for the first three features, the integration into Disp3D needed far more work than expected (2.1.3).
The needed amount of man days for Testing and Debugging turned out to be far below the estimate made during first phase. Because a lot of time was invested into extensive code review and documentation, there only were some minor bugs which could be easily fixed.

#### 4.3.2.6 Risk Estimate

As listed in the Risk Estimate (2.1.3), the project contained some degree of hidden complexity (R6). Especially the sensor data tree item was revealed to be far more complicated than expected (2.1.2). This resulted out of insufficient communication with the product owner, respectively an effort estimate that was provided by the latter but not reflected by the team. The hidden complexity of this feature was resolved by rearranging working groups.

# 5 Glossary

**Bad channels** are sensor inputs that do not deliver working signals or are not set up correctly.

**Compute shaders** are a shader type in OpenGL. They allow for custom computations to be executed outside the typical rendering pipeline.

**Disp3D** is a part of the MNE-CPP framework that holds 3D visualization functionality.

**Eigen** is a C++ template library for linear algebra that includes matrices, vectors, numerical solvers, and related algorithms

**Fibonacci Heap** is a data structure for priority queue operations, consisting of a collection of heap-ordered trees.

**GPU** (graphics processing unit) is a programmable logic chip specialized for display functions.

**Linear search** or sequential search is a method for finding a target value within a list.

**MNE-CPP** is a cross-platform C++ framework, which provides MEG/EEG tools and applications for fast non-invasive brain monitoring.

**OpenGL** is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.

**Qt** is a cross-platform framework, mainly used for software applications and graphical user interfaces.

**SCDC** (surface constrained distance calculation) determines the shortest path between two points on a tessellated surface.

**k-D Tree** is a special type of binary search tree for organizing some number of points in k dimensions.

# 6 References

In this section are listed some of the most important references where the work was initially based on by the information helping the team determine a better work flow and understanding of the project as a whole.

**C++** Information for working with C++
`http://en.cppreference.com`

**Eigen** Information on how to work and compile the program on Eigen
`https://eigen.tuxfamily.org/dox/`

**Qt** Information regarding Qt Creator and its use
`https://doc.qt.io/qt-5/`

**MNE-CPP** Information about the project
urlhttp://doc.mne-cpp.org/

**UML-Diagramms** Information on Visual Paradigm for creating UML Diagrams
`https://www.visual-paradigm.com/`

**KD Tree** Building a balanced k-d Tree in O(kn logn) time
`https://www.google.de/url?sa=t&source=web&rct=j&url=http://jcgt.`
`org/published/0004/01/03/paper.pdf&ved=0ahUKEwjujpTLy-rUAhUQfFAKHT69BG8QFggfM`
`usg=AFQjCNGqWfqBlUcoTgtfbISsxhRz8E3GtA`
THE PROBLEM IN HERE IS THAT THE URL IS LONG AND LOOKS TERRI-
BLE

**Jira Attlasian** Introducing how to work with Jira
`https://marketplace.atlassian.com/`