POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

On a Computationally Empowered Virtual Reality System for Real Time Intracranial Neuronavigation

Supervisors:
Prof. Francesco Paolo Andriulli

Candidate:
Alessandro Mascherin
ID: 242947

Academic year 2019-2020

Summary

Context

Physiological and electrical information about the brain can be obtained via a wide array of techniques such as the Electroencephalography (EEG), the Positron Emission Tomography (PET), Magnetic Resonance Imaging (MRI) and Functional MRI (fMRI) which all have their trade-offs.

One of the most popular acquisition techniques used both for research and for clinical application is the EEG which is an affordable and noninvasive technology that measures the electric potential on the scalp of the subject by means of a set of electrodes. The EEG is also used for EEG Source Imaging (ESI) which aims at computing the intracranial electrical activity from the electric potential measured on the surface of the head. The source imaging problem can be split into two key sub-problems, the forward problem (FP) and the inverse problem (IP). The forward problem is a computationally complex problem, whose goal is to map the electrical currents inside the brain to the readings acquired on the head surface with an EEG. The IP tries performs the opposite mapping and yields the position and intensity of the electrical activity inside the human brain based on the EEG measurements on the scalp surface. Numerical methods can solve the FP, typically boundary or finite element methods, and its accuracy only depends on the accuracy of the model and anatomical information of the subject. The IP, however, is an ill-posed problem whose solution is not unique and generally unstable.

The brain anatomical structure is commonly derived from MRIs with a procedure called brain segmentation. This procedure can produce 2D or 3D reconstruction of the human brain, which is essential in medical diagnoses or surgical planning.

Goals

The goal of this Thesis is the design of an immersive virtual reality (VR) system, capable of offering a real-time navigation inside the electrical activity of the human brain, by leveraging neuroimaging and electromagnetic source imaging techniques. This application has to display realistic brain structures, that can be freely explored by the end user of the system. One of the challenges tackled in this thesis is the smooth, fluid, and computationally effective integration of advanced state of the art source imaging techniques for allowing a new virtual reality neuronavigation environment. This has been achieved by engineering computationally empowered algorithms designed to solve the inverse problem. The brain activity displayed to the end-user of the virtual reality system is computed from a real-time ESI-inverted EEG reading or from pre-recorded data which can be of interest both in research and medical contexts.

Personal Contribution and Results

The different challenges encountered in this thesis have been addressed in two phases: a first phase focused on developing the data-processing pipelines used for recovering the intracranial activity from the EEG measurements and a second phase in which a virtual reality environment has been develop to present the reconstructed data to the end user in the most intuitive and anatomically correct way possible.

First, a significant part of the thesis has been devoted to the design and implementation of the software pipelines, summarized in the Figure 2.7, and tools essential for satisfying the requirements of an immersive real-time neuroimaging experience. One of the key problems we have tackled has been maintaining stable performances even when dealing with high-density research EEG, to remain compatible with real-time visualization. The 3D meshes of the brain and the white matter fiber tracts used in the project come from an elaboration of real human fMRIs, produced with some of the most recent brain segmentation approaches, to ensure that the anatomical modelling is accurate and reflects the state of the art in brain imaging.

Another main part of this thesis has been the development of several features into the virtual reality application Neurosurf. For this development, we used the Unity3D engine and its XR software development kit.

We have designed an user experience for both a virtual reality environment and a traditional desktop setup, capable of providing meaningful tools and User Interface (UI). The brain structures were created by making use of real human data adapted for the virtual reality visualization and the source imaging pipeline. We implemented a visualization system capable of displaying the potential generated by the brain activation on the meshes' surfaces, through an application of specific shaders. The neuro-navigation in the virtual environment was developed by exploiting an advanced motion capture tracking system that has been integrated into the project.

The final application has been used in several demonstrations, using both live-recorded data and simulations. It has been tested with various EEG configurations and different head models. By leveraging on the modularity of the pipelines built for this thesis, the application was always able to provide a fluid experience while maintaining a real-time elaboration of the EEG signals.

The virtual reality environment we have obtained can be adopted for teaching, research, or medical purposes. One of the possible applications can be the neurofeedback which is a therapeutic procedure that allows patients to regulate their brain activity by allowing them to visualize and navigate it in real-time.

Contents

Acknowledgements

Firstly, I would like to express my special thanks to my supervisor, Prof. Francesco P.Andriulli, for the opportunity to work on this project and for the support he has given to me in these months. With him, I would also thank all the PhD students, Adrien, Davide, Maxime, Clément and all the others, for the time they have dedicated to me.

Vorrei inoltre ringraziare tutti i miei amici, a partire da quelli storici del mio paese, dove nonostante la distanza, mi hanno supportato e sopportato per tutti questi anni. Un abbraccio enorme lo dedico a tutti i ragazzi della Nobile Magistrale: affrontare questo percorso assieme a voi è stata la cosa migliore che mi potesse capitare.

Infine un grande e sentito ringraziamento alla mia famiglia, senza il loro supporto non sarei mai potuto arrivare fino a qui: grazie per aver sempre creduto in me e avermi aiutato a diventare quello che sono oggi, sono fiero di voi.

Chapter 1

Introduction

This chapter provides a brief introduction to the background knowledge and notation used the in this work. It presents a brief description of the project in the context of existing neuronavigation technologies and their application.

1.1 Project Description

The human mind structure and functioning mechanisms have always been a fascinating topic for everybody, from scientists and doctors to non-experts. In recent years, the advancement in neuroscience and neuroimaging technologies has opened new frontiers in the visualization and analysis of the human brain, some of which will be presented in this thesis. Since the beginning of the 20^{th} century, innovative techniques to investigate the brain function and its electrical activity has begun to emerge. In 1924, the first human Electroencephalography (EEG) was recorded, giving birth to a new way to study the human mind activity [19]. Just a few years later, in 1936, the EEG began to be adopted for clinical use, and today it is still one of the most common instruments used in research and medical environments.

In the last decades, other technologies have joined the EEG in the field of the exploration of the human brain. The technological progress allowed neuroscientists to explore even further the operations of the human brain, leveraging on modern instruments like the Positron Emission Tomography

(PET) or Magnetic Resonance Imaging (MRI). The increased available computational power has also moved the scientist a step further: identifying the sources that generate EEG signals with a technique called EEG source localization. Nowadays, the information related to how the human brain works are vast, ranging from the chemical structure of the neuronal cells to the global electromagnetic model of the head. However, a lot of the inner mechanism are still shrouded in mystery and are still actively researched.

The idea at the root of this thesis project is to use of the most advanced virtual reality or augmented reality technologies that have spread in recent years, to display the result of modern neuroscience techniques. By making use of an immersive environment, it is possible to provide new alternatives in the brain exploration and diagnostic. Leveraging on state-of-art EEG source imaging technique it is possible to display, in real-time, the electrical activity of the human brain in a way that can be exploited for example from therapist to monitor the patient brain activity. This scenario can also provide additional insight on the inner mechanism of the brain activity to scientists and researchers.

1.2 Neurofeedback

One of the possible target for a new Virtual Reality application for an intracranial navigation of the brain activity is the neurofeedback [35]. The neurofeedback is a branch of the biofeedback, the fields that aims to teach the control of the body function by giving the patient a direct feedback on the body's information. EEG neurofeedback consists of the methodology used to teach a patient to self-regulate its brain function by showing the electrical activity recorded through an EEG headset with video and sound feedback. The three main areas of application for the neurofeedback are: (i) as a therapeutic tool (ii) as peak-performance trainer (iii) as experimental method.

All of these applications rely on the same basic project structure. The data are acquired through an EEG headset, or with tools such as Magnetoencephalography (MEG) or Near-infrared spectroscopy (NIRS). The data are then processed and adapted to provide direct feedback to the user, based on the intended goal of the neurofeedback session. A 2004 study [8] showed that the application of an immersive VR environment in a neurofeedback

session tends to obtain better results than the traditional neurofeedback techniques. The current VR technology, combined the state-of-the-art EEG source imaging techniques, can provide an immersive and realistic simulation of the human brain activities that can be effective in neurofeedback applications.

The application developed in this thesis focus on offering the most accurate information that can be extracted from an EEG, to provide precise feedback to the user. A direct access to the human brain with a real-time visualization of the intracranial activity can be used effectively by therapists to employ modern neurofeedback techniques.

1.3 State of the art

The adoption of modern XR technologies in neuroscience research is already underway with several different proposed integration. The availability and the power of the modern VR device allow a broader range of application, that can be combined with some of the more recent discoveries in the field. The adoption of virtual reality in Brain-Computer Interfaces (BCI), with games and ad-hoc peripherals is a promising field with several studies that supports its diffusion [34, 52, 9]. These researches often focus on adopting the XR technologies to facilitate communication with the BCI by providing direct feedback or other useful visualization to the user. For example, augmented reality was also studied for the adoption in BCI application [5], for the intuitive interaction and the possibility of having a "hand-free" experience. The visualization of the brain structure in a modern VR environment was also an object of study for different projects.

In 2002 a collaborative team composed of neurosurgeons from the Massachusetts General Hospital, researcher from Brown University and the National Institutes of Health, tried to develop an immersive environment for the exploration of Diffusion Tensor Imaging (DTI) images [48]. By making use of an immersive virtual reality environment like the CAVE [1] they developed a visualization system of some geometries derived from DTI MRI acquisition. The hardware limitation of a virtual reality system of almost

[1] `http://www.visbox.com`

20 years ago, however, generated serious visualization issues in the frame rate and the latency of the system.

The advancement in the VR technologies allowed years later, in 2016, a research group of the Seoul Korea University to import the results of a DTI and a conventional brain MRI in a Unity VR-based application [29]. The application they propose, shows a static representation of the human brain structure, and the navigation is exclusively done with the mouse. This project occurred in some limitation in the computational power needed to manipulate the DTI data correctly, and it was missing the motion capture functionality needed for a complete VR experience.

In 2014 the GlassBrain project offered the first immersive experience for intracranial navigation while displaying a real-time estimation of the source-localized brain electrical activity[47, 40]. By implementing advanced source imaging techniques, combined with a careful optimization of the GPU pipeline, they were able to display in real-time the electrical activity of the brain on a Unity application, with data coming from a high-quality EEG. GlassBrain project was used as a reference for this project for how to visualize the brain structures activity and how to provide valuable navigation information to the user.

Chapter 2

Technical Background

This chapter is dedicated to the explanation of some of the technical concepts needed for understanding the implementation of this project. The first section presents some fundamental brain physiology concepts, for then deal briefly with the neuroimaging and source imaging problem. At the end of the chapter, there is a section used to explain the tools used in this thesis, and some of the pipelines and algorithms adopted in the preliminary project phases.

2.1 Brain Physiology

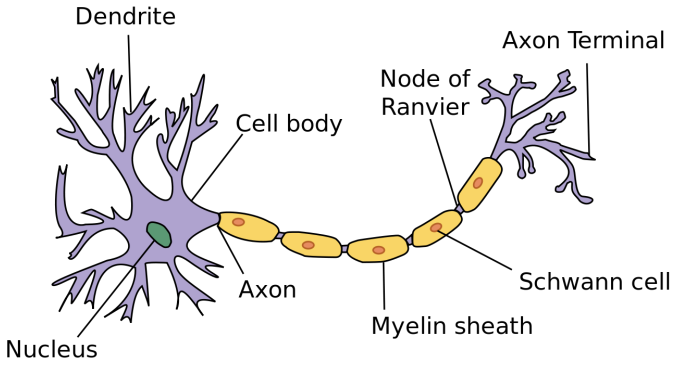


Figure 2.1: Structure of a neuronal cell

The human brain is one of the most sophisticated structures in nature. It has been estimated that it is composed of 10^{10} to 10^{11} individual neuronal cells [46]. Each cell is capable of creating a connection with thousands of other neurons, generating up to 10^{14} synaptic connection. The role of the neurons is to rapidly process and transmit information.

Figure 2.1 represent the main components of the neuron. The essential parts that are of interest in the study of the electrical activity of the neuronal cells are the axon terminal and the dendrites. The dendrites represent an extension of the cell body used to amplify the receptive surface of the cells. The dendrite's surface is covered with synapses, the point of functional contact between the neurons with other neuronal cells. The synapses connect with the axon of other cells, and they are responsible for transmitting signals received through the axons with an electrochemical process. The ability of neurons to generate impulses is based on a chemical operation based on the movement of ions of sodium (Na^+), potassium (K^+), and chloride (Cl^-). At rest, the neurons possess a negative resting potential of -70 mV. The signal is generated by a chemical substance called neurotransmitter, which, at a synaptic level, generates a depolarization and a potential difference. This depolarization is called **EPSP: excitatory postsynaptic potential**. The diffusion of the signal is stopped by another type of neurotransmitter, responsible for hyperpolarizing the inter-neuronal volumes. This phenomenon also generates a potential difference, called **inhibitory postsynaptic potential (IPSP)**. At the cell body, the contribution of the ESPS and IPSP received by the dendrites are summed together. When the EPSP potential sum generates a depolarization under the -55mV threshold, an impulse is generated by the cell and propagated along the axon to other neurons. These impulses are called "action potential" or "spikes." They usually have a voltage change size of about 90-100 mV, and they have a duration of few milliseconds.

The electrical activity that can be detected on the head surface cannot represent one of a single neuron since the activity of the neighboring cells covers it. The EEG signals are then extracted when a considerable number of cells activates simultaneously. These cells will generate an electrical activity that can be modeled as a current dipole.

2.2 Neuroimaging

Neuroimaging is a discipline dedicated to the acquisition and reconstruction of the human brain anatomy and function [1]. What characterized the neuroimaging from the techniques used in the past, is the commitment of techniques that allows an "in vivo" imaging, meaning an acquisition made on a living organism. This discipline can be classified in two main imaging categories: **structural imaging** and **functional imaging** [12]. The structural imaging goal is the brain structure and anatomy reconstruction. The conventional structural imaging techniques consist of magnetic resonance (MRI) and tomography (CT). This approach is applied to brain damage or abnormality diagnosis. Moreover, it can be used to obtain geometric properties of the brain structure.

The functional imaging, instead, focus on the activity analysis in the various brain area. Its role is the extraction of information related to brain function and connectivity. The activity is typically obtained through functional MRI (fMRI), positron emission tomography (PET), near-infrared spectroscopy (NIRS), and others. Also EEG is often used to scan the electrical activity of the brain. In this section, the technologies adopted for the head model creation and the brain activity measurement will be briefly introduced.

2.2.1 MRI, fMRI

The magnetic resonance imaging consist in a technique that makes use of magnetic fields and radio waves to generate images of the organs in the body [17, 22]. Its first clinical introduction was back in 1980 in Nottingham and Aberdeen and has rapidly evolved as one of the most powerful structural neuroimaging techniques. The MRI is based on the application of two strong perpendicular magnetic fields. The spacial resolution of an MRI is directly connected to the intensity of the magnetic field applied. The first static field $\overrightarrow{\boldsymbol{B0}}$ is used to align the hydrogen nucleus existing inside the body. The rotation along the field happens with a specific frequency called **Larmor Frequency** [51], proportional to the static field applied. To measure the magnetization generated by the magnetic moment of the hydrogen nuclei a second field $\overrightarrow{\boldsymbol{B1}}$, perpendicular to the first one is applied. This second field is usually applied with short pulses of few microseconds and causes

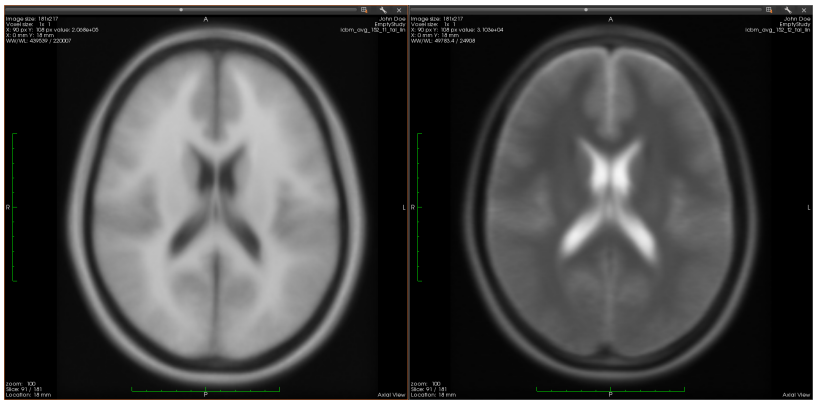


Figure 2.2: On the left: T1 weighted MRI.
On the right: T2 weighted MRI.
MRI taken from the NIST ICBM database.

an energy absorption by the nuclei. This energy is then emitted with a relaxation process and can be detected and measured. The relaxation can be longitudinal and transverse, given form to two different time relaxations constant, T1 and T2.

The magnetic image generated can be weighted for the different constant, to identify specific area. For example, water has a high T1 value and tends to appear dark in T1-weighted images. At the same time, fat has a short T1 time, looking brighter on the same T1-weighted MRI. This effect can be easily observed in fig. 2.2, where the Cerebrospinal Fluid (CSF) identified by an "X" shape in the center of the MRI, is almost black in the T1-weighted image while resulting bright on the same MRI T2-weighted.

The functional MRI (fMRI) is an imaging method, derived from the MRI, that aims to detect the brain activity based on the change in the blood flow. Seiji Ogawa discovered that the blood has different magnetic properties based on the oxygenation level, in a method called **BOLD (Blood Oxygenation Level Dependent)** [41]. By acquiring the MRI images at a higher frequency than the BOLD signal, it is possible to obtain a functional image of the activity of the brain based on the oxygen use of the neurons [25]. Images obtained with T1 and T2 MRI, and fMRI were extensively used in this project for the generation of the meshes and the tractography

used in the virtual reality environment. T1-weighted MRI were also used to obtain a labeled version of the human brain, as it is described in the section 4.2.

2.2.2 EEG

Electroencephalography is the most common method used to acquire the electrical activity of the brain. It consists of a set of electrodes placed on the scalp of a subject, with dry or wet connection. The wet-EEG makes use of an electrolyte placed between the skin to reduce the acquisition impedance level. Due to the low cost of the setup and the fact that is usually a non-invasive technique, EEG is commonly used in diagnostic (epilepsy, sleep analysis, brain death detection and others) and research (neuroscience, psychology).

The signals detected by each electrode consist of the sum of the voltage contribution of several neurons groups the activate simultaneously. This voltage measured on the surface has a low magnitude, usually ranging from 5 to 200 µV, and a low-frequency band (0.5 - 50 Hz). Moreover, the surface EEG has a high amount of noise due to the high resistance of the skill and several artifacts that affect the measurements. To overcome the noise problem, an invasive EEG methodology is also available, the intracranial EEG (iEEG). This methodology is not considered in this project.

The typical EEG setup consists of 20 to 30 electrodes, usually placed according to the standard '10-20 system' that defines the position and the nomenclature of a set of 21 electrodes on the patient scalp [18]. For research purpose, and for increasing the EEG spatial resolution, high-density version of EEG setup are used. With these systems, it is possible to have up to 256 electrodes on the scalp surface.

2.3 EEG Source Imaging

Trying to understand which brain region is active in response to certain specific mental tasks, is research of particular interest in the diagnostic and Brain-Computer Interfaces (BCI) fields. The neuroimaging techniques previously covered, try to extract this information with the best resolution possible. However, techniques such as the MRI, the fMRI, and the Positron-Emission Tomography (PET), despite having an excellent spatial resolution,

are not able to identify the temporal dynamic of the brain activity. To acquire information related to the neurons activation, it is necessary to have instruments with excellent temporal resolution, with a sub-second scale.

The EEG is one of the most common tools adopted to acquire the data with the required temporal resolution. The most significant limitation in the localization of the activation is given by the poor spatial resolution of the EEG [7]. In human patients, the EEG is usually recorded through electrodes placed on the scalp surface. This cause a resolution reduction due to the different resistive layer that the electrical signal generated by the brain activity has to cross.

These limitations, combined with the increased available computation power, led the scientist to the creation of a new research field called **EEG source imaging (ESI)** [37, 36].

The ESI research field focus in the solution of two main problems:

- **forward problem (FP)**: the goal of the forward problem is to establish the potential at the electrodes starting from a current density inside the brain;
- **inverse problem (FP)**: as the name suggests, this problem tries to solve the opposite of the FP, finding the source activation in the brain based on the potential read by an electrode.

Modern ESI techniques were used in this thesis to allow a real-time mapping of the EEG potential to the brain structure visualized in the VR application. The rest of this section will focus on a general FP and IP formulation while presenting some of the state-of-the-art solutions available.

2.3.1 Forward Problem

As mentioned, the forward problem focus on the resolution of the potential on the scalp surface based on an electrical activation in a specific brain area [36, 21]. The brain activity is usually modeled as a current dipole, defined by its moment

$$d = \|d\| \cdot \hat{n_d} \tag{2.1}$$

where $d = \|d\|$ is the dipole magnitude and $\hat{n_d}$ is the dipole orientation.

Assuming the dipole is located in a homogeneous isotropic material with conductivity σ, the potential field measured at generic point placed at a

distance r from a dipole placed at r_{dip} is given by the equation

$$g(r, r_{dip}, d) = d \cdot \frac{r - r_{dip}}{4\pi\sigma \left\| r - r_{dip} \right\|^3}. \tag{2.2}$$

If the dipole is located at the origin of the Cartesian coordinate system, aligned along the z-axis, the equation 2.2 became

$$g(r,0,de_z) = \frac{d\cos\theta}{4\pi\sigma r^2}, \tag{2.3}$$

where σ represent the angle between the vector $\mathbf{r}$ and the z-axis. This formulation exposes that a quadratic factor attenuates the dipole field.

The potential on each electrode position can be computed with a superposition effect, by summing the contribution of each dipole

$$V(\mathbf{r}) = \sum_i g(r, r_{dip,i}, d_i). \tag{2.4}$$

This leads to a generic algebraic formulation of the forward problem, that, for N electrodes, p dipoles and T time samples, can be expressed as follows:

$$V = \begin{bmatrix} V(r_1,1) & \dots & V(r_1,T) \\ \vdots & \ddots & \vdots \\ V(r_N,1) & \dots & V(r_N,T) \end{bmatrix} = G(r_j, r_{dip,i}, \hat{n_{d,i}}) \begin{bmatrix} d_{1,1} & \dots & d_{1,T} \\ \vdots & \ddots & \vdots \\ d_{p,1} & \dots & d_{p,T} \end{bmatrix} = GD \tag{2.5}$$

V is the matrix of potential measured for every time instant at every electrode position. G is the gain matrix and D models the dipole magnitude at every time instant.

The gain matrix G is also known as the **Leadfield Matrix**, and it is used to model the physical properties of the head. By also considering the presence of noise in the EEG reading we should include a noise component in the equation. This is done summing a Gaussian noise component represented by the matrix $\mathbf{n}$, leading the equation

$$\mathbf{V = GD + n}. \tag{2.6}$$

Numerical solution: FEM and BEM

The presented formulation is based on the consideration that the head is a homogeneous and isotropic sphere. Recent studies have shown that by employing a realistic model of the human head, the result obtained by the forward model are greatly influenced. The classical solution used to solve the FP are **Boundary Element Method (BEM)** and **Finite Element Method (FEM)**. The both are numerical methods that solve respectively integral equations and differential equations and can mode the brain electrical behavior. Details on the formulations for the BEM and FEM theory will not be discussed in this project since they are out of the scope for this thesis. For more details a good reference is [20]. The goal of both methods exploit segmented mesh of the brain elements to discretize the equation. Both FEM and BEM are also used to generate a leadfield matrix G used in the formulation of the IP. For this thesis, we implemented a BEM solution for the leadfield matrix generation and the creation of the head model.

2.3.2 Inverse Problem

The Inverse Problem is the logical opposite of the Forward Problems. Given the electrode potential, it aims to identify the dipoles that have generated such potential inside the brain. The IP is, however, a *ill-posed* problem: the number of electrodes used range from 21 to 256 sources, while the dipole sources are several order of magnitude more numerous since possibly every neuron can be modeled as a dipole source. Moreover, the number of unknown values (3 coordinates for the position and 3 for the orientation of each dipole) is bigger than the available system equations; for this reason, the solution is not unique, and it is unstable, presenting high variations in response to little input change.

There are two main solution for the Inverse Problem [16], based on non-parametric and parametric approaches:

- **Distributed Inverse Solutions (DIS)**: is a non-parametric solution, based on choosing several dipoles along the whole cortical, while imposing a fixed location and, possibly, orientation. For every chosen dipole, the FP is solved to build the correspondent Leadfield Matrix; By applying an inverse operation on the obtained matrix, with the addition of some constraint used to overcome the ill-posed nature of

the problem, it is possible to find a solution to the IP. This solution is based on the constraint applied and the FP model used in the lead-field generation. Common state-of-the-art DIS solution includes the **Minimum Norm (MN)** solution, the **Weighted Minimum Norm (WMN)**, **LORETA** [43] and **LAURA** [15];

- **Equivalent dipole methods**: set of parametric solutions that aim to find the best dipole position and orientation that lead to the best configuration compared to the EEG potential. The techniques have various range of complexity, with single or multiple dipoles and with spherical or realistic head models.

The DIS are the solution best suited for a real-time elaboration since the problem can be solved with a linear complexity solution. To obtain the required real-time visualization on a complex virtual reality system, we developed the model and the pipeline based on a distributed solution.

2.4 Project organization

A general overview of the hardware and software setup adopted on this work will now be presented. At first, in the section 2.4.1 the tools adopted are going to be described, followed by the software used and the adopted pipeline, in the sections 2.4.2 and 2.4.3 respectively.

2.4.1 Hardware setup

This section describes the hardware component used for the project, focusing on the workstation required to make it work and the virtual reality setup used.

Main Workstation

The main computer used for this thesis was a Windows 10 workstation with the following hardware specification: an Intel Core I7-8700 CPU @ 3,20GHz, 16GB ram and an NVIDIA GeForce GTX 1070 GPU.

Most of the computational power required from this project comes from two main elements: the Neurosurf Virtual Reality application and the ESI conversion pipeline that extracts the data from an electroencephalography

(EEG), elaborates it and sends it to the VR program. The EEG data can be simulated or can be acquired from a real-time EEG reading. In the case of simple simulated data or an EEG setup with a low number of electrodes, this workstation was powerful enough to run both the virtual reality application and the computational pipeline that sends the brain activity data to the Neurosurf scene. However, with more complex simulation or with a higher density of electrodes, a bottleneck was found in the system CPU, requiring a different workstation to elaborate the EEG signals. The EEG conversion pipeline used most of the CPU power. When dealing with a simulation with 14 electrodes, the pipelines uses around 40% of the CPU time used after the initial setup. The image 2.3 displays a brief recorded power usage snapshot for the script with said configuration. After an initial usage spike due to the setup phase of the program, the CPU stabilizes around the 50-55% threshold, resulting in a 40% net usage increase. Also, in prolonged usage of the program, the computation power required has shown to be constant.

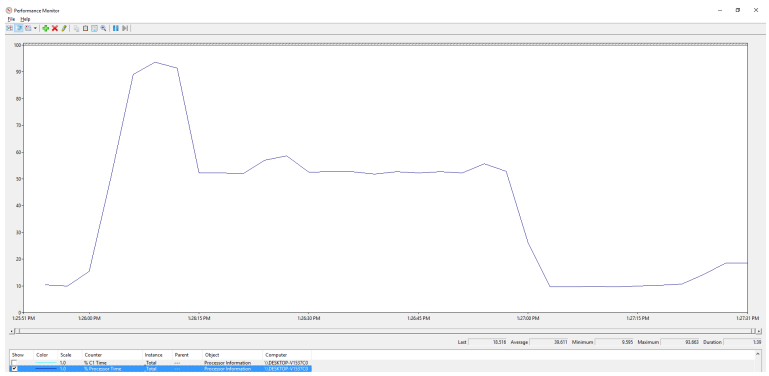


Figure 2.3: CPU Usage for the EEG conversion, 14 electrodes

When using more complex EEG system, the presented workstation was not able to run the VR application and the conversion pipeline simultaneously. For this reason, we have built the pipeline in a modular way, while focusing on advancing and optimizing the algorithms used. More details on this pipeline are going to be presented in the section 2.4.3. The virtual reality application, instead, relies mostly on the GPU of the workstation. On a normal execution, the Neurosurf application utilizes between 30 to

50% of the GPU power and from 10 to 25% of the CPU.

Virtual Reality Headset

Figure 2.4: HTC Vive Components, courtesy of HTC Corporation

The Virtual Reality visualization was done through an HTC VIVE VR system. The HTC VIVE is a consumer virtual reality headset, officially released on the market on the 7t June 2016. The VIVE VR setup includes a Virtual Reality Headset with a refresh rate if 90Hz and 110° of Field Of View. The user can interact with the system with two proprietary controllers, and the tracking is done with a "room-scale" technology called "VIVE Lighthouse." To enable this positional tracking Technology, HTC has developed the Base Stations, small black boxes that should be placed at the edge of the tracked environment. The tracking is done with an "inside-out" method. Each base station acts as an infra-red beacon, with two additional laser emitters. The headset recognizes and uses the signal sent from the stations to determine its position in the space with 6DOF, in relation to the location of the base stations. However, for this project, not all the provided hardware was used, since the tracking system was replaced by the OptiTrack tracking technology, that will be described in the following section. The VIVE controllers were replaced with a couple of customized PlayStation Move VR controllers, provided by OptiTrack. Only one of the provided Base Station was installed since the VIVE Lighthouse technology

was not used in this project. This base station is necessary since, if the headset does not detect at least one base station, it will cut off the video feed.

OptiTrack position tracking

Figure 2.5: Our OptiTrack camera setup, at Politecnico di Torino

All the positional information required for the virtual reality application presented in this thesis were acquired using OptiTrack. OptiTrack is a 3D motion data capture system that offers a high precision tracking, sub-millimeter with the right condition and calibration, and low system latency. The OptiTrack system works with multiple active 2D Infra-Red camera, installed around a target volume where the capture will take place. The cameras are synchronized by a control application that calculates the 3D position of the marker recognized in the recorded environment via triangulation. The marker can be both passive, simple retro-reflecting spheres or bands, or active, with LED markers that should emit in the 850nm wavelength. For this project, the headset and the controller were tracked using a configuration of several passive markers. The cameras consist of 8 OptiTrack PRIME 13W camera, installed on a metal cage that measures 5m x 5m with the cameras installed at 3m high. Each camera has a resolution of 1.3MP with a vertical Field of View of 58° and horizontal of 70°. The cameras can also acquire the environment using a grey-scale acquisition with a 750nm light wavelength filter. The maximum frame-rate possible

for each camera is 240 FPS, which is almost 3 times the frame rate of the VR headset used (90 fps for the HTC VIVE). This high frame-rate allows Motive to precisely calculate the position of the tracked object, even with fast movement, providing an accurate flux of information to the Neurosurf application. The maximum theoretical precision of the system, stated by the company is sub-millimeter. After the calibration phase of the system, our setup reached a mean error of 0.8618 mm in the position of each sphere. This value is subject to change over time, since every few months a new calibration needs to be made again and also depends on the room illumination.

gTec EEG Research headset

For the EEG signal acquisition we adopted a set of high precision research EEG headset provided by G.Tec medical engineering.[1] Most of the acquisition was done with the gNautilus, a wireless external wet-EEG with 32 electrodes. More sophisticated devices were also used, like a 256-electrodes passive headset, shown in figure 2.6 or a 64 active electrodes wet-EEG system. The signal recorded through this devices can be directly analyzed or stored in matrices by using the software suite provided by gTec. For the integration of these systems with the pipeline developed for this thesis, we implemented a small C++ interface used to stream the data acquired from the headset with the Lab Streaming Layer (LSL). The LSL protocol will be described in detail in the section 3.4.3.

2.4.2 Software used

This section will describe the key programs used for the realization of this thesis.

Unity Engine

Unity is an editor used to develop 2D and 3D application, offering a complete tool-set to programmers and artists. Its first release was back in 2005, announced as a MAC OS-X exclusive game engine. Nowadays, Unity has

[1] `http://www.gtec.at`

Figure 2.6: gTec 256 electrodes EEG cap used by our laboratory

evolved into a complex multi-platform engine widely used in the creation of games, films and animation, automotive and manufacturing applications or 3D engineering software.

Unity is built on a C++ core, with a cross-platform native structure. The scripting is offered through a C# API, which is then converted in the native C++ core with IL2CPP. IL2CPP (Intermediate Language To C++) is a scripting backend, developed by Unity, that converts the C# in C++ assemblies for the creation of the binary files. IL2CPP allows the final program to take advantage of the native C++ core, increasing the overall performance. Some of the core element of the game engine are:

- **The rendering pipeline** The Unity built-in render pipeline is called High-Definition Render Pipeline (HDPR). It is a pipeline developed for high-performance workstations and consoles, designed to produce realistic lighting and visual effect. It is based con Computer Shader technology and requires a Graphic Process Unit (GPU) able to process it to perform the rendering. The developers can optimize the pipeline for specific platform and hardware. With Unity 2018.1, the Lightweight Render Pipeline was also introduced. This pipeline has the goal of obtaining real-time render by making some trade-off in the lighting and the shading. It is a single pass forward-render pipeline, that is exposed through a C# API to the programmer, allowing direct

control of each render step;

- **The User Interface System** The engine provides tools to build User Interfaces for both the run-time application and the Unity Editor. The Interfaces for the editor are used to create a tool for the developers when building the application; The run-time UI is based on the GameObject systems, the base entity used to build all the Unity scenes.

- **The Physics engines** The physics engine is used to handle collision, acceleration, and physical simulated interaction between the object in the scene. It supports the NVIDIA PhysX engine and Data-Oriented Technology Stack (DOTS) based on Havok.

Unity is the market leader for the production of Virtual Reality and Augmented Reality content. In 2017, Unity announced a unified API to develop all XR content. XR is a term referring to all the applications that can be classified as Augmented Reality (AR), Virtual Reality (VR) or Mixed Reality (MR). It also implements a common single-pass stereo render pipeline, that can be adapted for multiple XR application. Unity Engine and the XR API were extensively used in the realization of the Neurosurf Virtual Reality application presented in this thesis.

Motive

Motive is a software platform provided by OptiTrack to record and manage motion capture data for different tracking application. Interacting with the OptiTrack Prime cameras, Motive can perform motion capture, in real-time, for objects with up to 6 DoF. Motive offers a complete suite of features and controls to achieve precise and complex tracking that can be recorded or live-streamed to other applications. It provides complete access to the tracking camera, allowing full control of the acquired images and accurate frame synchronization. It is possible to use one of the many preset to quickly setup a tracking environment or, if needed, further adapt and calibrate the system. The calibration phase is essential, like in many other motion capture systems. This phase is used to identify and build a 3D tracking volume, taking into account each camera position and image distortion. The calibration is done by moving a unique calibration wand in the tracked area until the system has acquired a sufficient number of samples. Motive than

elaborates the data and provides a report on the quality of the calibration, containing important information such as the overall error values and the suggested maximum tracking distance for each camera. Once the camera calibration is completed, Motive needs to setup the system origin and the position of the ground plane. This operation is done by using a convention of three markers, shaped in an L form. This marker must be placed on the ground at the center of the room and are used by Motive to complete the creation of the 3D tracked space.

With a calibrated system, Motive is able to recognize up to 2000 markers simultaneously. These markers can be grouped in Rigid Bodies. A Rigid Body is a data structure used by Motive to track an object with 6 DoF. To correctly identify the object, at least 3 visible markers are needed, up to a maximum of 20 markers per rigid body. The marker can be a passive reflective element or active sensor. If the markers are placed in unique positions, while avoiding geometrical congruency, Motive provides stable and precise information regarding its location and position in the 3D tracked space. This setup can also be used to record human body movement accurately. By placing markers on precise anatomical location, Motive can perform a full-body tracking and accurate movement analysis.

In the contest of this project, Motive and OptiTrack were extensively used to track the position and the orientation of all the real objects visualized in the virtual reality environment. It was also used to measure the position of the EEG electrodes on the patient head, as will be described in the chapter 5.

Paraview

ParaView is an open-source scientific visualization application. It is available for the most common desktop operating system, Windows, Linux, and macOS. It also offers several other framework and programming language APIs. For web or remote visualization, there are ParaView Python and ParaViewWeb, a complete Java-Script library. It can also be used in High-Performance Computing, in custom C++ application or streamed for immersive application with VRPN (Virtual-Reality Peripheral Network) interface. The user interfaces are written with the Qt open-source widget toolkit, while the data manipulation and rendering are done with Visualization Toolkit (VTK). The desktop applications are the most frequently

used interfaces. ParaView allows showing a wide variety of scientific data with various degrees of complexity, ranging from small data-set to more complex scientific information. This data can be loaded from a wide variety of file formats, and are then converted by the application in the versatile VTK format. VTK can effectively represent scientific data and meshes, allowing the users to visualize it in one of the several view-ports offered by ParaView. This data can be manipulated, filtered, and queried, using the tool provided by ParaView or from additional plugins that can be integrated with the application.

In the context of this thesis, ParaView was used to acquire the medical imaging used to build the neuro-navigation system and to convert some of the meshes in formats usable by the later stage of the established pipelines.

2.4.3 Pipelines

In this section some of the key preliminary pipeline will be introduced. To be specific, the pipelines used to generate the meshes for the brain and the white matter fiber are introduced. The pipeline used to transmit the EEG data to the neurosurfer application will also be presented.

EEG to Unity data transmission Pipeline

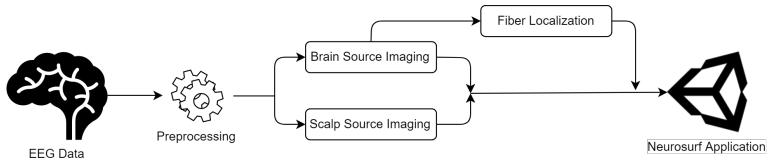


Figure 2.7: EEG data elaboration pipeline

The figure 2.7 outline the general pipeline developed to acquire the data from an electroencephalography or a simulation and the subsequent step implemented to stream the data into the Unity Virtual Reality scene. When acquiring a real brain electrical activity, the first operation that has to be done, it is the application of a prepossessing stage. This phase needs to clean the data, applying a band-pass filter to remove frequencies that are not useful in the later elaboration stage. It also scales the EEG values for the later stages of the pipeline. Then, the same data stream is processed by

three other stages. Each stage elaborates the data for a specific portion of the human brain: the scalp, the brain cortex surface, and the brain fibers.

The scalp and the brain source imaging stages take the potential measured by the EEG electrodes and elaborate it for the final visualization. The pipeline has to output an intensity value for each vertex of the meshes displayed in Unity. The details of these implementations will be presented in the section 3.3.1 The fiber localization step has to identify which fibers are activated based on the recording received. The brain fibers displayed in the Neurosurf application are a simplification, done for a visualization purpose, of the whole white matter tracts existing in the brain. For this reason, the algorithm identifies the group of fibers in the Neurosurf application that correspond to the real brain element that has generated the signal. The input of this pipeline stage is the potential values found on the surface of the brain mesh by the previous source imaging step.

Brain mesh Generation Pipeline

The human head mesh was obtained by making use of publicly available human MRI and fMRI. Some of the datasets used in this project were the ICMB-NY[2] and atlases provided by the NeuroImaging & Surgical Technologies (NIST) Lab. The first step of the pipeline adopted was the segmentation of the MRI to obtain a 3D geometric mesh. This operation was done with Freesurfer [3].

FreeSurfer is a public platform that provides a set of tools for processing and the analysis of human brain MRI and fMRI images. By using the FreeSurfer tool-set, we were able to generate a mesh for the scalp, the skull, and different brain models. The brain models were then processed by Brainstorm [45], an open-source application dedicated to the elaboration of magnetoencephalography (MEG) and electroencephalography (EEG). Brainstorm allows an integration of the EEG source imaging map, needed in the context of an ESI pipeline, with the result of the FreeSurfer elaboration. Brainstorm allows an alignment of all the data in the same coordinate system, based on the subject physical characteristic. The Electrodes can then

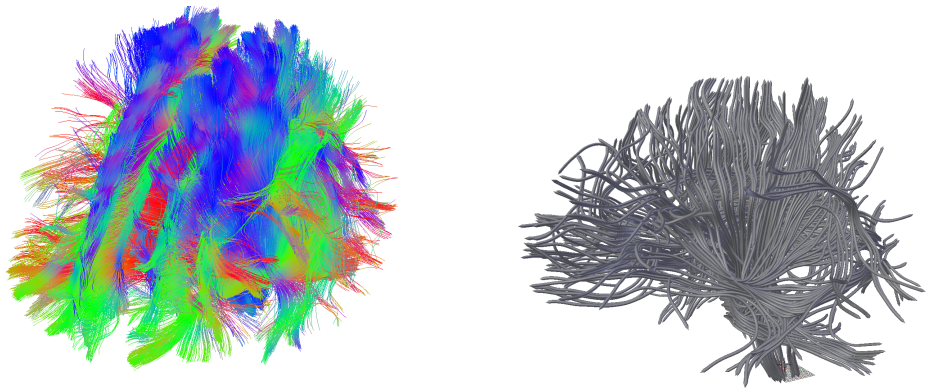
[2] `https://www.parralab.org/nyhead/`

[3] `http://surfer.nmr.mgh.harvard.edu`

be placed on the reconstructed head geometry, for then process these pieces of information to generate the leadfield matrices needed for the Forward and Inverse Problem.

The import in Unity was done through Paraview. The output of Freesurfer was converted in a VTK format and subsequently converted in a format compatible with Unity.

Fiber Generation Pipeline



(a) medInria tractography *(b) Blender conversion*

Figure 2.8: Comparison between the initial tractography and the fiber pipeline result.

The brain fibers, also called association fibers, consist of axons that connect specific cortical areas inside the brain. Obtaining the whole connectome of the human brain is a challenging operation, of particular interest in neuroscience applications. The creation of a white matter tract map is usually done by a process called tractography. The tractography is a DTI (Diffusion Tensor Imaging) 3D modeling technique that aims to estimate the brain connectome starting from diffusion magnetic resonance imaging (dMRI) [26, 27].

A tractography consist in a large dataset of streamlines, sequence of point located in the 3D space, that can be presented using 2D or 3D visualization. However, a tractography can be made of a high number of streamlines, with an order of magnitude of about 10^6 elements, making the

interpretation and the elaboration of the connectome a problematic task. Since the final goal of this pipeline is the generation of a set of meshes that will be used in a complex Virtual Reality environment, a clustering algorithm was implemented to reduce the number of images obtained from a tractography. Starting from a set of dMRI, a tractography can be extracted and visualized with medical image processing software such as medInria.

The obtained tractography has then been processed with the QuickBundles algorithm [13], an unsupervised learning algorithm able to extract centroid streamlines with a complexity that is linear in respect of the number of input streamlines. With QuickBundle, we were able to extract about 400 centroids from the starting tractography generated by medInria. The obtained centroid and streamlines were used to generate a Leadfield matrix for the fiber weaving, to apply the EEG source imaging techniques from an EEG recording.

To create a mesh that can be imported and visualized in Unity, we have established a python based pipeline for the elaboration of the centroids. The first step of the pipeline consists of a recentering operation of the points, to match the reference system with the one used by Unity. Then, by using the Blender python engine, the mesh of the fiber were generated, by using the algorithm wich pseudo-code is presented in the algorithm 1. Blender is an accessible free and open-source 3D creation suite. Its core is written in Python and can be used to generate 3D geometries algorithmically. This algorithm presents the operations done on the data obtained by QuickBundle to create a mesh compatible with Unity for every fiber. Each fiber is then exported in the .fbx format. This set of meshes is then imported in Unity together with the file listing the centroids position, used to identify each fiber in the final scene.

```
Data: list of centroid and fiber lines extracted from QuickBundles
Result: A 3D mesh for every input centroid and lines
for every centroid do
    Extract the termination point for the centroids;
    for every line do
        if lines belongs to centroid group then
            Add it to a centroidvertexlist;
        end
    end
end
for every element in centroidvertexlist do
    Create a Spline curve that passes through the vertex in the list;
    Add a Bevel modifier;
    Fill the generated curve;
end
```

Algorithm 1: Fiber creation algorithm

Chapter 3

Virtual Reality System

This chapter describes the proposed implementation for the virtual reality system used for real-time neuronavigation. First, the project organization will be covered before focusing on some of the most important features developed and how they impact the overall application. Some implementation details are presented at the end of this chapter.

3.1 Neurosurf Introduction

The goal of this project is the creation of a new virtual reality intracranial neuronavigation system able to provide a real-time visualization of the human brain electrical activity. This environment offers a navigable and realistic 3D representation of the human head, displaying its main structures such as the scalp, the brain and a visualization of the brain fibers. These structures have to correctly represent the real-time activation of the various brain areas deduced from either recorded or simulated EEG recordings. State of the art EEG source imaging techniques has been implemented to map, in real-time, the electric potential read by the sensors on the surface of the scalp to the brain structure visualized in the proposed application.

Several constraints must be satisfied to achieve the proposed objective. The EEG source imaging problem should be solved in real-time, to obtain a correct visual representation of the electrical signals object of study. This problem requires computationally expensive operations. For this reason, we have developed advanced algorithms capable of solving the source imaging

problem with a minimum delay, allowing a real-time visualization of the recorded electrical activity. In addition to the source imaging problem, a virtual reality environment demands a high amount of computational resources. Every frame should be rendered at least two times, three in case of a simultaneous desktop visualization. The integration of a VR application with the presented source imaging algorithms is a delicate task that has required a careful optimization since a fluid navigation is a fundamental component of a virtual reality experience. The motion sickness that can be caused by a virtual environment is a known issue [23], that is still a challenge with the modern VR technologies, especially when dealing with computationally heavy setups. To solve these challenges we have designed and implemented several solutions that are presented in this chapter.

The application was developed using the Unity Engine, with most of the programming elements written in C#.

3.2 Project organization

Unity organizes all the objects, environments, and menus needed for an application in a structure called a scene. As a comparison with a traditional computer game, the scene can be seen as a container for all the elements needed for a single level. Unity defines every object present in the scene as a GameObject. The GameObject is an entity that can model any element of the application such as meshes, lights, or event scripting logic. The properties of a GameObject are called *components*, which can be specific property provided by the engine (like the transform component for the position) or scripting component created via C#. In the proposed work, we have organized the virtual reality application in a single scene containing all the main elements (Figure 3.1).

3.2.1 Camera

A single stereoscopic camera, called *[CameraRig]*, is used for both the virtual reality experience and desktop visualization. We have attached to this object the scripts responsible for the tracking management and the camera position in the space. The camera, in the VR environment, acts as the "eyes" of the user in the virtual world. When positional tracking is enabled the

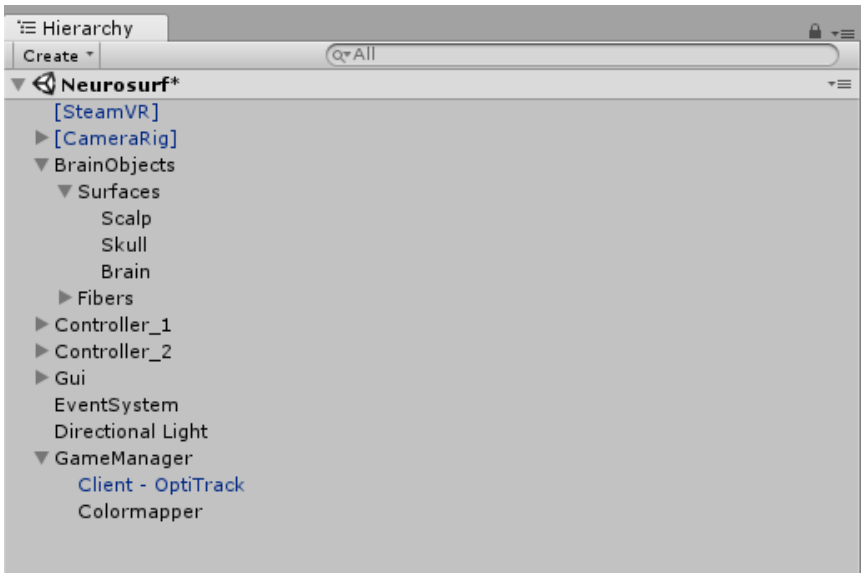


Figure 3.1: Neurosurf's scene hierarchy

camera position responds only to the user movement. The camera movement in a virtual reality environment should appear natural to the user, and avoid modifying the camera status without user interaction.

The camera component is responsible for the definition of the rendering viewport dimension and the position of the clipping plane in the scenes. The clipping planes are two imaginary planes set in front of the camera to define the rendering frustum, the area, that should be rendered and projected on the 2D screen. The "near" clipping plane was placed very close to the camera position. This placement was chosen for allowing a mesh display when the user is navigating inside the brain structure thus avoiding unpleasant visual effects like geometries that disappear when the camera comes close to the object. When using the application in a desktop configuration, without the headset, only one of the two images captured by the stereoscopic camera is displayed and rendered on screen. The camera, in this configuration, is controlled by the user input on the keyboard.

The chosen stereo-rendering for the HTC Vive headset is the single-pass option offered by Unity. On a traditional rendering pipeline for a stereoscopic camera, the scene is iterated twice, and the rendering pipeline

is independent for the left and the right eye. This process in Unity is slightly optimized since it requires only a single pass of the scene, and the culling and shadow computation steps are shared for the eyes. However, to optimize an XR application, Unity introduced the single-pass rendering. With this mode, the rendering pipeline is executed only once per frame, by rendering both eyes at the same time. Between each draw call the render pipeline switch the viewport between the two eyes, alternating the object rendering between the two eyes. This operation is faster than the multi-pass camera since, despite the overhead introduced by the viewport swap, it allows a single traversal of the render pipeline [2].

3.2.2 Illumination

The illumination is limited to a single directional light set to cast no shadow. This light is primarily used to give the user a depth sensation when observing the various brain tissue and the brain fiber weaving. Most of the illumination in the scene comes from the brain activity and the fiber activation. The activation is presented with a change on each mesh vertex color and with light emission from the most active one. The dark background helps making the brain activity stand out, adding a sense of immersion to the scene. The use of a 3D virtual room in which the user can move while immersed in the virtual reality was tested, but it resulted in being confusing and reduced the focus of the user to the brain activity. The other lighting and texture needed in the creation of a realistic environment reduced the ability to observe the data displayed in the brain model. To keep the observer's attention on the neuro-navigation, no meshes other than the brain structure are included in the scene.

3.2.3 C# scripting component

An application manager GameObject, the *GameManager*, was used to control the global status of the scene. We have implemented the GameManager following the singleton pattern. Its primary role is the management of some status variables and of the system configuration. By creating a singleton class, we have made the important variable accessible in other sections of the code like, for example, the transparency values that can be changed by the user from the UI. Other parameters are exposed in the editor for quick

adjustment by the developer. In addition, the GameManager is responsible for the connection setup with OptiTrack, with the LSL streaming used in the source imaging pipeline and with the tool adopted for managing the controller. The colormap manager was inserted as a child of the GameObject. The colormap manager behavior is dependent on the data received by the streams, made available by the stream activated by the GameManager. The *Gui* GameObjects manage the User Interface. This component contains the design of the UI displayed in VR and desktop mode, and the C# scripts used to control the user inputs through the UI.

3.2.4 3D polygonal meshes

All the brain-related 3D geometrical elements are organized hierarchically. An empty parent object, *BrainObjects*, is used to control the global position and orientation of the meshes in the space.

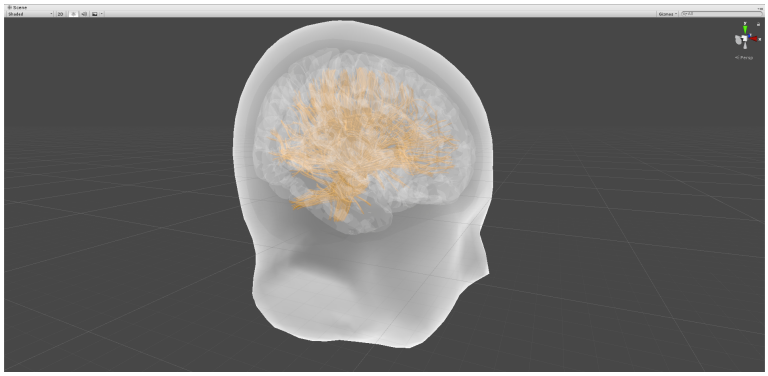


Figure 3.2: Brain meshes as shown in the Unity editor

Brain meshes and structure

The brain meshes consist of several elements, as can be seen in Figure 3.2:

- An external semi-transparent scalp. This mesh is relatively simple, consisting of only 1082 vertexes and 2160 triangles. The scalp will display a simple outline of a generic human face, providing positional information such as the location of the eyes or the ears.

- Inside the scalp there is a mesh representing the skull. This mesh is built with a low number of vertexes (642 vertexes and 1280 triangles), since there are no electrical activities that will be displayed on it.
- The brain mesh, instead, is the key element of the proposed visualization system. Various meshes were adopted for different simulations, with different degrees of complexity. The simplest mesh used was made with about 15,000 vertexes and 30,000 triangles. The system, however, is capable of supporting more complex 3D structure, going up to 1,500,000 vertex and 560,000 triangles, providing a much higher resolution and fidelity. The exploded mesh generated with the Mindboggle process that will be presented in Chapter 4 consist of over 130.000 vertexes, split in 70 independent sub-meshes.

The brain activation is displayed on the scalp and the brain, while the skull acts as an inert surface. The electrical activity recorded through the EEG electrodes is shown on the surface of the scalp. The scalp displays the potential on the surface of the head, based on the elaboration done by the source imaging pipeline. Each EEG reading corresponds to the activation of a part of the brain, displayed on the surface of the brain mesh. For each vertex of the mesh, in each frame, a potential value is computed by the source imaging pipeline and received by the application. This value is then applied with a colormap on the vertexes of the brain mesh. Each mesh has a specific shader attached used to display the information that each brain element should suggest to the user. The shader used for the skull is the simplest one since it does not need to show any brain activation. For this reason, the shader is made with a gray base color with an additional fresnel effect used to increase the sense of depth of the mesh. This shader supports transparency because an opaque mesh would completely hide the brain and fiber mesh underneath it. For the scalp and the brain instead, some specific vertex shader is used. The color of each vertex is based on the electrical intensity computed. The color is assigned according to a specific colormap, process described in detail in Section 3.3.2.

Brain Fiber

Another critical element displayed is the representation of the brain are the fibers contained inside the human brain. To achieve a more effective

visualization, only a condensed part of the complete geometric structure of the human brain fiber is displayed. In the scene, there are around 400 independent meshes, each representing a bundle of fibers contained in the brain. The creation process in the pipeline was previously described in Section 2.4.3.

All the fibers are grouped in a single object, containing a script used to handle all the individual fiber meshes during the simulation. This script, called FiberManager, reads from a textual file the coordinates of the centroids used to identify the fibers and to map the detected electrical activity correctly. When displaying the activity of the brain and the fiber, an LSL stream sends a level of activity associated with the fibers. According to a configurable parameter called the activation density, the script defines the size of the portion of the fibers that will be displayed as active. When a fiber is set to active, its shader is changed to display the recorded activation. The shader for the active fiber displays a flow animation, showing the direction of the electrical activation along the fiber. More details on the shader adopted are presented in Section 3.4.2.

Controllers Meshes

When using the application in virtual reality mode, the input is managed by a couple of PlayStation Move controllers. To make the interaction natural for the user, they need to be rendered and placed in the virtual environment based on their position in the real world. There are then two meshes that model the controller in the scene, only showed to the user when using the application with the virtual reality system.

3.3 Key Features

3.3.1 EEG source imaging implementation

This section describes the implementation of the already introduced source imaging pipelines.

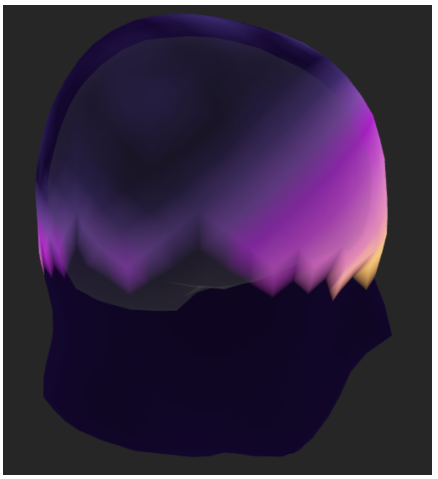


Figure 3.3: Scalp activation as shown in the VR scene.

Scalp Source Imaging

This pipeline has to elaborate the potential value on every vertex of the scalp mesh, based on the potential read by the EEG electrodes. The parameters used in the setup phase of this program are the vertex coordinates used for the scalp mesh and the coordinates of the electrodes. The coordinates are extracted from the mesh used in the ESI pipeline, with the electrodes placed on the model using a standard configuration. By using the program proposed in Chapter 5 it is possible to apply this pipeline to a generic electrode position setup on measured location position. With these parameters, the algorithm generates two interpolation matrix of order 4 by using an interpolation based on a spherical spline approach, proposed in [44]. These matrices, one for the scalp vertex and the second for the electrode position, are generated at the program launch, to reduce the computational power needed during the application of the interpolation to the sample. The procedure is made of three steps:

- The electrode position is placed on the sphere used to model the scalp, obtaining a set of spherical coordinates for each electrodes

- The potential data are interpolated on the sphere surface according

to the following relation

$$U(E) = c_0 + \sum_{i=1}^{n} c_i g(cos(E, E_i)) \tag{3.1}$$

where z_i are the potential value measured at the i^{th} electrodes, interpolated by the spherical splines U, E_i are the electrode spherical coordinate and c_i are the solution of the system $\begin{cases} GC + Tc_0 = Z \\ T'C = 0 \end{cases}$, with $T' = (1,1,\ldots,1)$, $C' = (c_1, c_2, \ldots, c_n)$, $Z' = (z_1, z_2, \ldots, z_n)$ and $G = (g_{ij}) = (g(cos(E_i, E_j)))$.

The function $g(x)$ is defined as

$$g(x) = \frac{1}{4\pi} \sum_{n=1}^{\infty} \frac{2n+1}{n^m (n+1)^m} P_n(x) \tag{3.2}$$

where P_n is the n^{th} degree Legendre polynomial. Based on the Perrin research, m was chosen with m=4, while n is equal to 7, for having a precision of 10^{-6} on $g(x)$.

- These splines values can then be projected on the surface plane, by using a radial projection.

For visualization purposes, we have limited the interpolation area to the upper part of the scalp, excluding from the elaboration every scalp vertex placed at a distance greater than a fixed tolerance from any electrode. The interpolated value for every mesh vertex is obtained by applying a scalar product between the vector of the EEG samples received from LSL and the interpolation matrix computed at the startup. The resulting vector is normalized and sent through an LSL outlet to the Unity VR application.

Brain Source Imaging

This step of the pipeline is responsible for the application of the ESI techniques described in Section 2.3. The brain leadfield is generated with a BEM technique, while the algorithm used for the resolution of the inverse problem is sLORETA [42]. Part of the optimization done for this algorithm

is a pre-computing phase at the start of the algorithm used to extract some intermediate matrix from the input leadfield to accelerate the elaboration loop used to process the data. All the matrix operations were performed using the Python NumPy library, that allows for an efficient vector representation and enables high performance in the operation [49].

Fiber localization

To identify which fiber is active based on the EEG reading on the scalp, we have split the problem in two parts. At first, we have created a Leadfield for the coordinates corresponding to the two end of the condensed fiber we have obtained through the QuickBundles process. This leadfield is used to map the brain current distribution to the potential at the end of every fiber structure used in the visualization. By using the result of the IP generated by the brain ESI phase, the fiber localization algorithm elaborates the potential at the two end of every fiber cylinder. If this potential difference is higher than a threshold (imposed to filter out noise and low activation values) the fiber is considered active and it is flagged for activation in the LSL output. For every fiber Unity receives an indication of the fiber status, activating or deactivating it based on the stream values.

3.3.2 Colormap

In the scientific data visualization, a major design choice that has to be made is how to color the data. It is necessary to define a function that maps numerical value with a range of color. The range of color chosen is commonly known as "colormap." A colormap is then a sequence of continuous color, distributed along a range of values. The reason to use a colormap is to provide to a human observer the ability to rapidly identify the areas that represent the values of interest. The choice of the right color set is essential for transmitting the right information and give the observer the most effective clues to achieve a correct interpretation of the scientific data visualization.

Some color patterns, like the commonly used rainbow patter, despite the use of a large set of color, have been proven to perform poorly by multiple studies [6, 39] Another factor that must be taken into account when choosing an effective colormap is the different sensibility of each observer and,

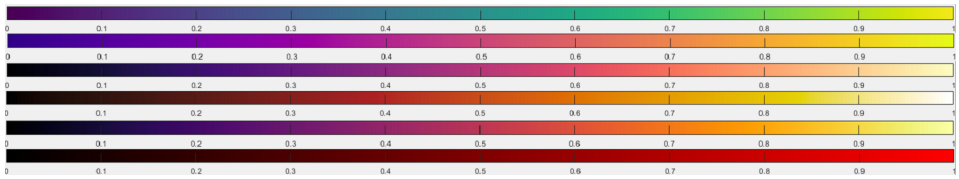


Figure 3.4: Some of the colormap used for the project. From top to bottom: viridis, plasma, magma, black body, inferno, blacktored.

possibly, the variety of display system that could be used. In this project the chosen colormaps are displayed on both a traditional LCD monitor and on the AMOLED display used by the HTC VIVE.

The adopted solution was to provide the user the capability of choosing a colormap between a set of pre-selected maps. The colormap patterns were taken from some of the colormap contained in the Python Matplotlib Library and from the map proposed by Kenneth Moreland on its web-page [38]. The colormap change can also be easily applied during the execution of the application.

Mapping between values and color

Every chosen colormap has been converted into a .CSV file containing 256 lines of RGB values, corresponding to the color in the map in the various range. In C# we have defined a *ColorMap* class, used to model each colormap, providing useful helper methods. The main field of this class is a list of colors, mapping the values extracted from the input .CSV file into the color Unity object. The color object allows the specification of an alpha value for each color, that can be applied to a texture if the shader supports the transparency. To emphasize the higher values of the colormap, we have added a linear transparency option. By specifying a minimum and a maximum alpha factor, the transparency range is linearly subdivided over the 256 color value existing in the ColorMap.

Another class, called *ColormapperHandler* was used to apply the colormap to the mesh vertices and to change the visualized colormap at runtime rapidly. This class was implemented using the singleton pattern, to avoid multiple allocations of the list of instantiated colormaps and to provide a single point of access to the colormaps information in the application.

This handler keeps track of which colormap is actually in use, exposing the colormap information of the active map. It also permits the change of the linear transparency threshold for every stored colormap. This swap is an expensive operation, that is not available to the final user, but it was exposed as a property in the Unity editor for fast adjustment during the setup phase of the system.

The actual mapping between the data received from the source imaging pipeline and the colormap stored in the ColormapperHandler class is done by a script attached as a component to the relevant objects. The data received by the pipeline are stored in a queue since they arrive at an irregular rate. Every frame a new vector of values is pulled from the top of the queue. These values consist of floating-point numbers representing the electrical activity on each vertex of the brain and the scalp. This data are normalized according to the maximum and minimum received value and then scaled along the colormap.

```
1  private ColormapperHandler cm;
2  ...
3  //data received by the LSL inlet
4  values = (double[])receiver.datasetQueue.Dequeue();
5
6  //colors is an array of Colors with a size equal to the mesh
       vertex number
7  cm.ApplyColormap(values, colors);
8
9  //the color matrix for the mesh is replaced with the one
       extracted by the colormapper
10 mesh.colors = colors;
```

3.3.3 Interaction mode

The application was developed with two distinct configurations to provide the best user experience in both a virtual reality system and a traditional desktop setup. The visual information that the system provides is the same in both configurations, but it is presented to the user in different ways. To accommodate the use of these two configurations several features were adapted. The camera is positioned differently to frame the 3D model in a meaningful way in the two environments. The input used is also different: in a desktop setup the user can use the mouse and keyboard, while in the

virtual reality system the user uses a couple of controllers. With different input systems comes the need to change the user interface, particularly the option menu.

Traditional Desktop Mode

In this mode, which we have called "Keyboard mode", the application is meant to be interacted with a keyboard. The scene can be seen using a traditional monitor or with the VIVE Headset. When the system is in this configuration, the tracking for the headset is disabled and the headset acts only as a static immersive display. The brain is positioned at the center of the screen, and the camera is always pointed towards it, by making use of the *LookAt()* function provided by the Unity engine. The user can interact with the scene using the keyboard with the following key binding:

- The camera can be rotated with the keys W, A, S, and D. With W and S, the camera rotates along around the X-axis of a positive or a negative value, respectively. The A and D, instead, rotate the camera around the Y-axis. The rotation is done using the center of the scene as a pivot point.
- Using the keys O and L, the user can "zoom in" or "zoom-out". When pressing the corresponding key, the camera is moved along the axis that connects the camera position with the center of the screen.
- Pressing C changes the colormap used for the visualization, as described in the following section.
- By pressing M, the user can access the menu. The menu UI will pop-up, partially covering the scene and allows the user to change several settings. From the menu the user can choose which object to display and can choose a transparency value for each mesh. There is also a help section, meant to explain the key-bindings used in the project.
- With the TAB key, the user can switch to the virtual reality mode. This keyboard key remains also enabled when in VR if the user wants to come back to the desktop mode.

The camera movement is designed with a fixed-step approach. At each update a script checks if one of the mapped movement button is pressed.

Then the distance from the center, the horizontal angle and the vertical angle is updated with a value specified as a configuration parameter in the editor.

We set some boundaries to the possible camera movement, to keep the visualization a controlled process. The minimum allowed distance from the center was set to 0. The maximum allowed value was chosen to allow the minimum dimension of the visualized brain to be of a 1/10 of the screen width, in a 16:9 aspect ratio monitor. The far clipping plane of the camera frustum was placed at a distance such that the brain polygons are all covered at the maximum camera distance. The horizontal rotation can be done around the entire brain, with no limitation on the maximum angle. The vertical rotation was instead limited to a -90° to 90° boundary, offering a 180° arc of exploration.

Virtual Reality Mode

When the user is using the virtual reality headset, the camera position is based on the physical position of the headset, tracked by the OptiTrack cameras. Motive calculates the coordinates of the headset and the controllers and sends it to the Unity application. The controllers, disabled in the "keyboard mode," are enabled and placed on the scene on the location identified by Motive. If a controller position is not detected the controller is hidden from the user.

In the virtual reality environment, the brain is positioned at the center of the virtual room where the user can explore and move around. The tracked area that can be reached by the user measure around 9 square meters, spanning along the cage used for the tracking. To give a more natural user experience, the zoom in/out functionality offered in the desktop configuration is replaced with a "drag and re-scale" interaction. If the user presses both of the triggers on the controller and brings the controllers closer, the meshes are made smaller. The opposite happens when moving the controller away from each other. With the same key combination the brain can be moved alongside the vertical y-axis. This gesture is obtained by moving both controllers up or down, simultaneously. The same shift is then replicated on the mesh. The maximum and minimum height reachable has been limited with parameters we have set beforehand. With the analog trigger, the user can move the geometries around the x-z plane, the plane

parallel to the room floor.

The menu was also changed compared to the desktop mode. A 2D flat surface, positioned at the center of the field of view that occludes most of the vision in a 3D environment is not an acceptable solution since it can cause a severe discomfort sensation and the interaction is not natural. Instead, the menu is positioned far away from the user, in a location where the user can still easily read the writing and can interact with using the controller. An adjustment was applied in the render order, to make sure that the menu appears in the foreground even if the user is inside the brain meshes.

Mode change implementation

To manage the swap between the two configurations, it was necessary to design two separate user interfaces. In Unity, all the UI elements are placed in an abstract place called Canvas. In the Canvas, it is possible to change the UI rendering property and the dimension of the user interface in the scene. To keep the UI of the two configurations independent, the adopted solution was to create two different Canvas, one for the VR scene and one for the desktop scene. The Canvases are then managed by a script that activates or deactivates the UI corresponding to the correct configuration. In a typical use case, when switching from the virtual reality mode to the desktop mode, the script hides the VR menu element and deactivates the VR canvas. It then activates the desktop canvas and enables the menus contained in the canvas.

The other important features that must be controlled between the two modes are the position of the camera and the objects in the scene and the way the camera position is updated. When in desktop mode, the camera position is no longer bound to the tracked headset position and is controlled with the keyboard. Moreover, the controller is not needed when using the keyboard so they can be deactivated and hidden from the scene, to avoid unnecessary rendering operations. To make this change, the meshes and the initial camera position in each configuration are temporarily saved at the program start. When in the desktop mode, the position tracking is disabled, and the keyboard control is enabled. The position and the scale of the meshes are restored, and the camera position is reset in the center of the scene, pointing at the objects. When switching to the VR scene, the camera is temporarily set at the center of the virtual room, until the tracking

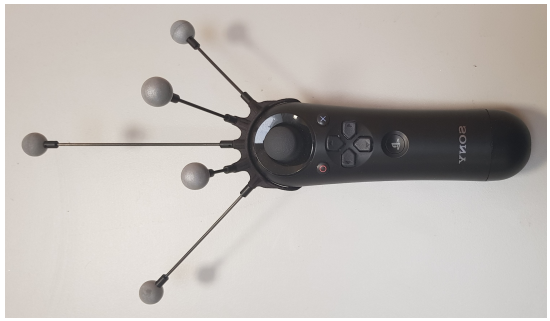
information reaches the application. If the tracking system is not on, or if there are some problems with the tracking information transmission, the user can still see the meshes, but it cannot move around the scene. This solution was adopted to prevent the erratic camera movement that can be caused by the absence of the tracking data, that can be very annoying for the user. The controller is then enabled, and the corresponding meshes are displayed.

3.4 Implementation details

3.4.1 Position tracking

Instead of using the HTC Vive's tracking system, the position tracking is done through the OptiTrack camera setup. OptiTrack provides a Unity plugin to allow access to all the information that are tracked with its cameras. This information is then provided to the Unity scene and are bound with the GameObjects that represent the various tracked object. A specific marker set was built to track the position of the VR headset. The marker is attached to the helmet through a custom 3D-printed support (Figure 3.5a).

(a) Headset marker setup.



(b) Controller marker setup.

Figure 3.5: Some of the markers pattern adopted for the VR hardware.

A significant challenge with the adopted setup was the management of the position and rotation tracking integrated with the HTC Vive headset. The headset shows the video feed only if it detects at least one of its base stations used by the system to track the position of the headset in the room.

The headset then updates its position in the virtual scene according to the tracking information obtained by the HTC Vive lighthouse. It is possible to deactivate the position tracking, but the rotational position in the 3D space of the headset is always active. Since it was not possible to override this setup, the adopted solution was to manually undo any update that the headset receives from the HTC Vive lighthouse. Between each frame, the program checks the updated position of the headset obtained trough the HTC lighthouse. This information is obtained through the Unity Engine XR API. With a 3D translation, the headset is then reverted to its original position, effectively negating the update done through the Lighthouse. After this step, the position information acquired through the OptiTrack system can be applied to the camera position. The following code snippet describes the applied transformation:

```
transformation.rotation = transformation.rotation *
    Quaternion.Inverse(InputTracking.GetLocalRotation(XRNode
    .CenterEye);
```

This script is attached to the camera object, and it is executed once per frame, in the Update() step of the UnityEngine pipeline. InputTracking is the part of the API used by Unity to interact with the tracking system of XR applications. The XRNode.CenterEye represent the point between the eyes that corresponds to the camera position in the scene.

3.4.2 Shader and Rendering

The light emission, the color, and the transparency of all the active geometries in the scene realized with the implementation of specific shaders. A shader consists of the algorithms and code that can be used to calculate the color of each pixel during the rendering phase. The input parameters of a shader are usually the lighting and the material adopted for the surface, but it can also include more complex user-defined values. The two main components of a complete shade are the vertex shader and the fragment shader. The vertex shader is applied on every vertex of the 3D geometry and is used to transform all the vertex information (color, position, texture, etc) from the original "object space" to the so-called "clip space". The clip space is the area used by the rasterizer to determine which pixel of the final image needs do to be drawn. The fragment shader is instead applied

"per-pixel". For every pixel that the object covers on screen, the fragment shader is responsible for the elaboration of the colorization, the depth and alpha testing, the texture application, the lighting and shadow, and other post-processing effects.

In Unity, the shader creation can be done by using the shader graph tool or by writing code in a variant of the high-level shading language (HLSL). The graph tool allows for a visual creation of the shader, while displaying a real-time preview of its effect. HSLS is a Microsoft proprietary language used for DirectX. It is written in a C-like language, called Cg (C stands for graphics). For this project, we have developed a specific shader for every 3D mesh that needs to display the electrical activity of the brain.

Brain Shader The brain is the most complex mesh in the scene. It consists of a high number of vertices and faces, placed in a non-convex configuration. This mesh should be rendered with a semitransparent effect, to display the fiber located inside it and the backside of the brain. The standard transparent shader offered by Unity, however, does not write into the depth buffer. During the culling phase of the rendering pipeline, if a complex non-convex mesh is not inserted in the z-buffer, it can result in drawing order problems.

The culling operation is an optimization process in the rendering pipeline that, after the application of the vertex shader, verifies which polygons are facing away from the camera, and removes them from the rendering pipeline. This operation is usually performed to avoid a rendering operation for all the polygons that are not visible for the user. The workaround we implemented is the filling of the depth-buffer before the transparency is rendered, allowing a correct culling operation while preserving the semi-transparency effect. The following code snippet illustrates the key parameter set in the shader to obtain the desired effect:

```
1 SubShader {
2     Tags {"Queue"="Transparent+1000" "IgnoreProjector"="True
          " "RenderType"="Transparent"}
3     LOD 200
4     Pass {
5         ZWrite On
6         Blend SrcAlpha OneMinusSrcAlpha
7         //in the first pass we render the back faces
8         Cull back
```

```
9           ...
10          //fragment and vertex shader are located here
11      }
12
13      Pass {
14          ZWrite On
15          Blend SrcAlpha OneMinusSrcAlpha
16          //Then we render the front faces
17          Cull front
18          ...
19          //fragment and vertex shader are located here
20      }
21  }
```

The rendering queue parameter is a value used by Unity to establish the order in which transparent and opaque objects are rendered. In this project, since we have three transparent objects (scalp, skull, brain) covering an opaque structure (the fibers), we had to specify the transparency rendering order explicitly, to achieve a correct visualization. The Projector is a Unity visual effect that allows a projection of a material onto all the object covered by the frustum of the projector object. This effect does not work effectively with a semitransparent object, so we have disabled it. The RenderType flags used to categorize the shader in a specific group, like opaque, transparent, background, and others. In the pass section of the shader, we force a write in the z-buffer but we split the operation for the two sides of the faces. Since the mesh is not convex, due to the presence of the two separate hemispheres and the sulci on the surface of the cortex, in the first pass we add to the buffer the polygons facing towards the viewer. These polygons correspond to the external brain surface, the gyri, when the user is viewing the mesh from outside and the sulci when the camera is immersed within the geometry. The second pass adds the remaining polygons, allowing a correct display of the various layer of the brain.

In the vertex shader, the color of each vertex is applied based on the color array generated by the colormap algorithm. The fragment shader, instead, applies the color on every pixel and is responsible for the application of the transparency value chosen by the user in the options menu:

```
1  col.a = col.a * Transparency
```

Skull and Scalp shader Skull and Scalp are simpler convex meshes than the brain. For this reason, we have adopted a Unity graph shader. The Albedo, the parameter that controls the base color of an object is based on a vertex color multiplied by a Fresnel effect. For the skull, the vertex color is unique while for the scalp the color array is generated by the color-mapper. We also added an emission and specular propriety, also based on the vertex color. The transparency is applied to the whole shader, and it is based on the alpha value set in the UI.

Fiber shader For the fiber, we developed two distinct shaders, one during the rest phase, and a second one for the activation. When the fiber is not active, the shader is a simple opaque shader, with an added Fresnel effect to add some reflective effect at the border of the fiber cylinder. For the activation, we implemented an animated shader, used to display the propagation of the electrical signal inside the fiber. We do not show the exact flow of the electrical signal, but instead, we display an animation on the fiber as long as electrical activity is registered on it. A preview of the shader effect can be seen in Figure 3.6. This solution was adopted since it is

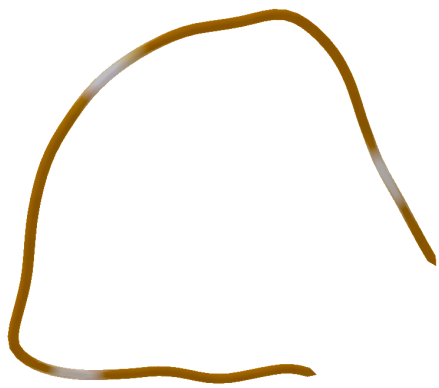


Figure 3.6: Fiber shader preview.

difficult to display the fast electrical transmission in the neurons effectively. The animation was done by moving a white-striped texture alongside the fiber, controlling the tiling and offset parameter of the Albedo UV. The tiling and the offset are changed by a distance determined by a simple

operation of *speed* × *velocity*, where *velocity* is the parameter we used to regulate how fast the white stripes should move alongside the fiber cylinder.

3.4.3 Stream management

The application receives all the inputs from several different streams. We have split the data flow used to display the electrical activity on the fiber, the brain, and the scalp in three different streams, implemented by using the Lab Streaming Layer (LSL) protocol. In VR, two additional input flow has been included: the positional tracking of the objects and the input received by the controllers. Both of these information are sent to the Neurosurf application by Motive, with a server-client paradigm.

Lab Streaming Layer (LSL)

The Lab Streaming Layer (LSL) [4] is a library designed to stream scientific data across devices over a shared network. It is composed of a core transport library (liblsl) available in several language interfaces (C, C++, Python, Java, C#, MATLAB). On top of liblsl, several tools can be adopted for acquiring data from the lab instrumentation (like, for example, an EEG), visualize or record it.

The LSL API provides some abstractions that can be implemented to transmit or receive scientific data:

Stream Outlets this data structure is used to push the data into the network. When an outlet is created, the stream is announced on the network, according to the network setup and layout, by using UDP multicast messages. When a client subscribes to the inlet, a TCP connection is established between the two hosts. The data can be pushed sample-by-sample or chunk-by-chunk, with a regular or a variable rate but with a uniform value type.

Resolve Function functions provided by the LSL API that can be used to identify which streams are currently active on the lab network.

Stream Inlet used to connect a client to an active streaming Outlet. From the outlet, it is possible to retrieve meta-data on the received stream and obtain the samples sent by the outlet.

All the streams are natively synchronized using the Network Time Protocol (NTP). LSL was used to implement the communication channels between each step of the EEG elaboration pipeline. LSL does not offer a secure transmission of the data since it was designed for controlled laboratory environments. If the security of the streams becomes an essential parameter for the application, a VPN-based solution should be implemented to protect the data transmission. LSL allows multiple subscriptions from the inlets to a single outlet. This feature has allowed the implementation of a modular pipeline in which the source imaging algorithms for the brain, the scalp, and the fibers can be split into three separate program, that can be executed by different workstation connected on the same network, overcoming computational limitation when using high-density EEG or complex brain structure. The data received by these three modules come from the same outlet generated by the prepossessing algorithm.

The Virtual Reality application receives the data from the various components of the pipeline, generally placed on other computers connected to the same network. Having the computationally heavy task parallelized on different workstation frees resources for the computer dedicated to the virtual reality rendering.

Motive C++ Wrapper

As already mentioned, the camera frames received by OptiTrack are elaborated by Motive to obtain the position of each recognized marker in space. Motive acts as a server, able to stream data using a UDP unicast or multicast protocol. For this project, the configuration we have adopted is based on a transmission on the local-host address, since we have utilized the same workstation to manage the tracking and the VR visualization. The frames acquired from the cameras are sent via UDP multicast through a dedicated Ethernet port in the main workstation. On top of the standard Motive application, OptiTrack offers a C/C++ API that can be used to implement the Motive functionalities without a graphical user interface. To obtain a complete integration between the tracking system and the Unity application, we have developed a C++ application using the Motive API, to acquire the tracking data. At launch, Unity starts the C++ application with a calibration file, a configuration file, and a target FPS parameter. The configuration and the calibration are generated in the standard Motive application and

then exported for the integrated C++ platform in an OptiTrack proprietary binary format. The calibration file contains all the information related to the camera position and orientation, the reference system adopted, the origin of the system and the location of the floor. The configuration file exports all the data related to the configured tracker set, the rigid body defined in Motive, the network configuration and more. Moreover, we have added a parameter to specify the target number of positional information that the application should provide at every frame, to match the framer-rate coming from the camera with the visualization frame-rate of the VR headset. With the HTC Vive, the maximum FPS supported are 90, but the tracking system can stream information with up to 240 frames per seconds. This C++ application, after loading the calibration and importing the configuration, activates the camera and starts sending the elaborated position to Unity.

SCP Server and XInput

The controllers used in this project consist of a modified version of the Sony PlayStation Move VR controller. The tracking sphere of the PS Move has been replaced by a set of five retrospective markers of 1.2 cm of diameter. When connected, the controllers act as standard Sony Playstation Controller. To acquire the input from a Sony controller on a Windows platform, a specific driver and a wrapper for XInput is needed. XInput is the standard API, available through the DirectX SDK, used to acquire the input of generic game controllers in a Windows environment. XInput is a C++ based API, but an open-source .NET wrapper is available, allowing the adoption of the Xinput functionality in Unity [14]. This wrapper was implemented in the Unity application to access the input data sent by Motive.

Motive interfaces with the Sony controller with a tool called ScpToolkit. ScpToolkit is a software composed of the Windows Driver needed to access the PS Move controller and an XInput wrapper to map the data to the standard DirectX configuration. The input is acquired by Motive and bundled with the positional information streamed via UDP. Each controller button set is mapped inside Motive with the corresponding set of markers organized in a RigidBody.

In Unity, the information regarding the input of each controller is acquired by interrogating the XInput API, using the index of the rigid body

as an ID (*playerindex* in the presented code):

```
GamePadState state = GamePad.GetState(playerIndex);
```

GamePadState is a struct offered by the XInputDotNet wrapper to map the controller inputs, that is built as follows:

Listing 3.1: XInput GamePad structure

```
1  namespace XInputDotNetPure
2  {
3      public struct GamePadState
4      {
5          public uint PacketNumber { get; }
6          public bool IsConnected { get; }
7          public GamePadButtons Buttons { get; }
8          public GamePadDPad DPad { get; }
9          public GamePadTriggers Triggers { get; }
10         public GamePadThumbSticks ThumbSticks { get; }
11     }
12     }
```

For every element of the struct, we implemented a C# event following the .NET Observer Design Pattern [3]. When a button is pressed the corresponding event is raised and every class subscribed to that event receives a notification. For example, by pressing the X button, the UI menu is shown. The UI class is, in fact, subscribed to the event related to the X Button and has a method to manage the received change in the button status.

Chapter 4

Brain segmentation

This chapter is dedicated to the presentation of a script designed to convert the brain segmentation result obtained in Paraview in a format compatible with the VR visualization. It will also present an additional feature developed for the system, which aims to give the user the opportunity to select isolated brain area, by leveraging on the Mindboggle software tool-set.

4.1 Brain Mesh conversion

The Visualization Toolkit (VTK) is one of the most common software used for the manipulation and visualization of 2D and 3D scientific data, in particular for medical imaging. It is an open-source software, written in C++, with wrapping for Python and Java. Almost all the medical 3D images used for this project were exported and computed using VTK. A game engine like Unity is not programmed to display scientific data and does not support a direct import of VTK 3D images. Some interaction between Unity and VTK were recently created, such as the possibility to render VTK medical imaging data using OpenGL directly in a Unity scene [50]. This solution was avoided because the brain element displayed in Unity was, in most cases, reworked to obtain a better visualization effect or for computational reason. Moreover, the project utilizes a combination of meshes that comes from both a .VTK format (the brain, the scalp, and the skull) and others that were modified in a .fbx object (the brain fibers).

As described in chapter 3, the brain electrical activity is displayed

through a change in real-time of the color of each mesh vertexes. To effectively apply this operation to all the elements in the scene, it has proven to be necessary to have all the meshes under a single render pipeline. As a result, the adopted solution was to convert all the .VTK in a format that can be read and manipulated by Unity. According to the Unity documentation for the used version of the software (2018.4), the engine supports Filmbox (.fbx), Collada (.dae), Autodesk 3ds Max (.3ds), Autocad (.dxf), and Wavefront (.obj) files. The solution that was adopted before this project was the conversion of the .VTK file in an intermediate format that can be opened with a third application, with format like PolyLine (.ply) or STereo Lithography interface format (.stl).

The program used for the final conversion step are typically some 3D computer graphics modeling software, like Blender o Autodesk 3ds Max; both application can generate an output mesh with a format that can be read and imported in Unity. The conversion sequence mentioned above is lengthy, requires the knowledge of multiple software and can visibly modify the mesh between each step of the pipeline.

The proposed solution was to develop a script to directly convert a .VTK file in one of the Unity supported formats. The data structure used by both the Wavefront and the VTK file are very similar, so the .obj format was chosen as the target for the conversion.

VTK File format The VTK files can be saved into two different formats: a legacy text-based format that can be directly read and modified, and a binary based on an XML file. The legacy files are written in ASCII and are subdivided in 5 parts:

- **Header** Header that identifies the VTK version used.
- **Title** Maximum 256 characters used to name the file
- **Data Type** This section can be either ASCII or binary. This line is also used in the XML version.
- **Dataset structure** Line used to describe the geometry and topology of the data contained in the file
- **Dataset attributes** The actual values of each element of the geometric figure saved in the file. Each data type starts with a line containing

a keyword describing the data (e.g., *VERTEXES*) and the number of lines associated with that data.

The XML file can be saved as a serial file, in which all the information is contained in a single file, or using a parallel structure. With the parallel structure, the dataset is split into several elements, and it is designed to work with multiple parallel processes used to read or write the data.

Wavefront OBJ File format The Wavefront .OBJ is an ASCII text-based format used to describe the geometry information of an object. It can also be written in a binary form to obtain better compression. The .obj describes a standard to represent polygonal geometry based on points, lines, and faces. Curves and surfaces are also supported, for the representation of free-form geometry. Each line in the text file can either be a keyword or a geometric value. For the values, the first ASCII characters in the line are keywords used to identify the information contained in the line. For example, the string

```
v 1.000 -1.000 2.000
```

is used to represent a vertex at the coordinates (1,-1,2).

The file has to be subdivided into sections. In each section, the keywords appendix has to be listed in an order defined by the format. Some of the elements that were used for the conversion, with the respective appendix are geometric vertexes (v), texture vertexes (vt), texture normal (vn), point (p), line (l) and face (f).

Implementation Most of the geometric information is saved in the same way in the two formats. The reference system is also the same, so it is possible to append the correct .obj appendix at each of the .vtk geometrical data to obtain a valid format conversion. Since the .VTK file can be also be written using a serial procedure, the VTK Python wrap was used to manipulate both the legacy text-based .VTK format and the more recent parallel file format. Python was chosen for the flexibility and the portability of the script and the possible integration with other application.

The script loads the VTK file and converts every geometrical information contained in an ASCII string that follows the OBJ standard.

The source code for the script is presented in the appendix 6.2.
The script can be applied over a single file or over a folder containing multiple files that need to be converted. Since the conversion is text manipulation and does not require complex geometric calculation, the operation has proven to be fast and reliable. With the meshes presented in the following 4.2 subsection, the elapsed time for the conversion of the entire brain was under 1 minute. The resulting .obj file retains the label name of the original .vtk file, a particularly vital information when several brain parts are split in different files.

The converted files can then be directly imported in Unity, without the need for further processing.

4.2 Brain Labeling and Mindboggle

The main component of the human brain, the cerebrum, is not a uniform structure. It is composed of two hemispheres that can each be further divided into specific lobes. Over the past decades, the scientific literature has defined several labeling protocols to precisely identify the different regions of the cortex. Anatomical brain labeling is an operation done on brain images acquired through Magnetic Resonance Imaging, and is used for a wide variety of applications, ranging from medical diagnostic to advanced brain analysis pipelines. Traditionally, the labeling procedure was done by hand, but it is an impractical and lengthy process. For this reason, several automated labeling algorithms were developed over the years [32] [11].

In the context of this thesis, the brain structure that is shown to the virtual reality system is not labeled, so it can not be used to specific analysis or comparisons between defined anatomical structures. The high-performance source imaging algorithms used to display the electrical activity of the brain on the mesh surface can be integrated with a labeling process, providing the opportunity of identifying the activation of different brain parcels.

4.2.1 Mindboggle

The Mindboggle project [30] consist of an open-source platform that aims to generate volumes, surface, and data of the human brain, taking in as input a set of preprocessed T1-weighted MRI data. The project started in 2005 as a doctoral dissertation. Today Mindboggle is available to the general public,

and it is also distributed as a cross-platform Docker container, easing the configuration and reproducibility of the platform. The software is openly available on GitHub and can also be found on their main website[1].

The Mindboggle project also offers one of the most extensive data set of manually labeled human brain [31], that can be used as an atlas for the creation of labeling platform. Mindboggle adopts the Desikan-Killiany-Tourville (DKT) Atlas protocol [10] for label classification. The DKT identifies 31 regions per hemisphere, but can also be adapted in a reduced variant of 25 regions per hemisphere. The Mindboggle processing algorithm can be synthesized as follows:

1. The T1-weighted MRI data, obtained from software like Freesurfer, are converted in .VTK surfaces

2. Mindboggle offers the option to combine the MRI with Advanced Normalization Tools (ANT) segmented volumes.

3. For each labeled regions obtained in the previous step, the software extracts the volumetric shape measure

4. The shape measure for every surface vertex is computed

5. The cortical surface features are extracted and segmented with labels from the DKT atlas

6. Additional measures for each label or sulcus are computed

7. The final step consist in the computation of statistics for every shape and collection of vertices

The output of this pipeline consists of several volumes, in NIfTI format, meshes, in VTK format, and tables. It provides labeled surfaces in the DKT protocol and surfaces that can be visualized in Paraview to extract information regarding the features of sulci and fundi or shape measures. The shape measures are also condensed in distinct label, feature and vertex tables.

[1] `https://mindboggle.info`

This platform was of particular interest for this project due to an additional optional output: an exploded version of the brain mesh, in which all labeled regions are split in individual mesh.

ROYGBIV

During the 2015 Brainhack hackathon, the Mindboggle team has developed a web-based visualization of the shape generated by the Mindboggle software. This project, called ROYGBIV [28] aimed to visualize the morphology of the human brain on a web platform, leveraging on the meshes obtained by the Mindboggle segmentation. For every cortex area, the application visualizes the shape measures and statistics computed by the Mindboggle pipeline.

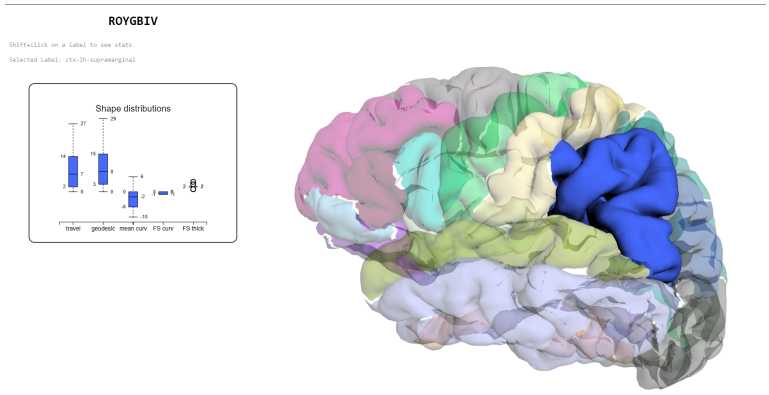


Figure 4.1: ROYGBIV Web Application

After studying this project, we decided to implement in the Neurosurf Virtual Reality application a similar feature. Mindboggle is able to generate an "exploded" version of the brain mesh, as displayed in the ROYGBIV application, that allows the user to select a specific brain label.

4.2.2 Neurosurf implementation

To implement the Mindboggle label segmentation in the Neurosurf project, we used the Docker bucket offered by the Mindboggle project, We have obtained the MRI data from the Connectome Database Project [24]. The

segmentation and labeling process, by using the Mindboggle platform in a single-core mode, took about 8 hours. The time can be reduced when deploying the Freesurfer segmentation operation in a multi-threaded configuration. All the output meshes obtained by this operation are saved in the .VTK format. By using the conversion script described previously at 4.1, we were able to convert all the meshes in a format supported by Unity. This meshes are composed of 70 separated geometries, one for each DKT label. The global brain mesh is formed with a total of 130.000 vertexes and 260.000 triangles. To manage all the meshes in Unity, we implemented a simple script used to apply a Unity Material to all the meshes child of a GameObject. This script was used to streamline the configuration phase when applying the texture to the whole brain.

One major problem we had to solve in the transition from the VTK format to Unity was the identification of the specific label name in Unity. When opened in Paraview, each label mesh has a tag to associate a numerical label identifier with the name in the DKT protocol. To recognize the labels, during the export phase, we set each mesh name to be the label ID used in Paraview. Then the association ID code - DKT name, was saved as a JSON file. At the launch of the Neurosurfer application, a script deserializes the JSON file and encapsulates the associations in a public, static, and read-only object. Each field of this object is a string named after the ID code, which value is the DKT name associated to that specific identifier. When another Neurosurf class needs to access the DKT name, the string value is obtained by using the C# reflection mechanism, as shown in the following snippet:

Listing 4.1: Reflection access to the labels name

```
1 //Using reflection to obtain the name of the label from the
      deserialized JSON
2 //The labels object is exposed through the GameManager
3 //The information can be accessed with info.GetValue(labels.
      name)
4
5 FieldInfo info = GameManager.instance.labels.name.GetType().
      GetField(mesh.name);
6 var labelName = info.GetValue(GameManager.instance.labels.
      name);
```

The selection of the label in the virtual reality environment was implemented with two different techniques: a laser-based interaction and a direct touch with the controller.

Laser based interaction

The laser pointer is an interaction technique commonly used in VR application. The user can project a laser for the controllers, interacting with the environment around him.

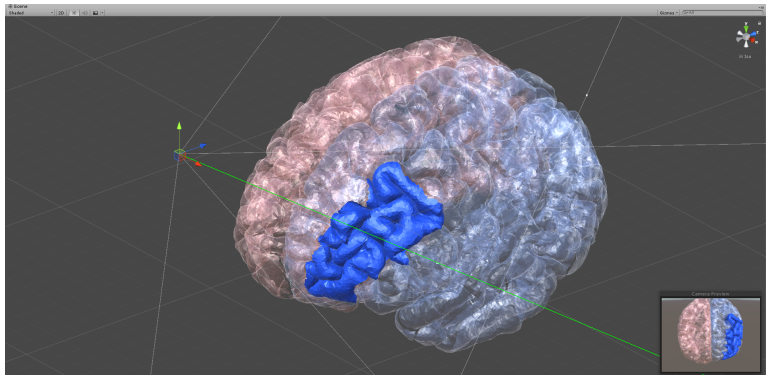


Figure 4.2: RayCast test during the development phase

The first implementation of the laser we tried was based on a ray-cast mechanism. By pressing a specific button on each controller, a Ray is cast starting from the controller position. Then, by doing a Physics Raycast check, the Unity engine is able to establish which brain label is hit by the ray. The ray is drawn by using the LineRenderer Unity component. This solution was also easily extended for the Desktop version of the Neurosurf application. In this configuration, the ray is cast by clicking the desired label on the screen. The Ray starts from the camera position and its cast on the screen in the direction extracted by using the Unity function:

```
Camera.ScreenPointToRay(mouse.position);
```

The problem with this specific implementation in the VR environment is caused by the immersive navigation that the user can perform inside the brain mesh. In this position, the use of the controller became uncomfortable, since the target meshes are surrounding the player, and the pointing

operation is difficult and un-intuitive. To avoid this problem, we introduced a touch interaction, based on the direct interaction of the controller with the mesh.

The laser interactivity, however, was not entirely discarded. The desktop version was kept intact since it was easy to use and to implement. The VR version was slightly changed. The RayCast check was removed, to avoid unnecessary computation since the label selection was implemented with the direct touch mechanism. The laser was transformed into a rigid object, used to point specific areas of the brain and to interact with the UI menu. The laser is a rigid geometry, that is scaled based on the hit point of a test ray. To avoid additional raycast query, that may impact the VR performance, the ray is cast only when the menu is active, and the check is done only for the layer used for the UI.

Touch based interaction

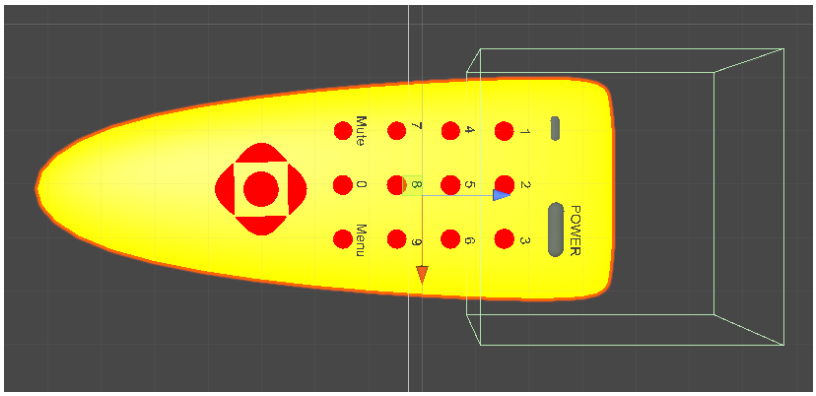


Figure 4.3: Controller Box collider

The touch-based interaction was implemented by using the Unity collider component. The collider defines the shape of an object for detecting physical collision. It is an invisible structure that approximates the object position, and it is only used in the physics detection pipeline. We added a static non-convex collider on every label mesh of the brain. The collider is not used for physical simulation, but only for hit detection, so even with the complex meshes adopted, the colliders were easily managed by the engine.

The idea of this interaction system is that the user can use the controller as virtual hands, and directly "touch" the surface of the label he is interested in. To implement the hit recognition on the label colliders, a box-shaped Trigger collider was added at the end of the controller, as shown in the picture 4.3. By using the OnTriggerEnter Unity event, raised when a trigger collider detects the collision with another collider, we implemented an Event-based system to manage the collision. The class subscribed to the event is responsible for highlighting the specific mesh touched by the user. This visualization effect is done by increasing the transparency of all the other brain mesh element while making the touched label completely opaque. The event system was implemented since, in future improvement of this project, other classes can subscribe to this event to start the visualization of specific information related to the selected label, like the name or some stats extracted from Mindboogle.

Chapter 5

Electrode Localization

This chapter presents a proposed method to acquire the position of EEG electrodes using an external motion capture system, like OptiTrack.

5.1 Problem Introduction

When trying to recreate an accurate electromagnetic model of a human head and brain, the exact dimension of the head and the geometrical location of the electrodes are fundamental parameters. If the adopted head model is derived from a standard template model instead of the patient-specific MRI, the spatial electrode coordinates can be used to adapt and correctly scale the dimensions of the brain structures and electrical fields estimation.

A high-density research EEG is composed of a large number of electrodes placed on the head of the patient. This high number of elements on the head's surface creates a challenge in the measure of the position of each electrode, since it is not possible to add additional marker or reference points on the EEG cap. A possible solution can be manual measurements or the adoption of Computer Vision techniques which, however, usually require expensive cameras set-up. Manual measurements are complex, time consuming and prone to errors. Ad-hoc measurement device based on the motion capture technology already exist on the market, like the Polhemus

device. [1] With this device, the electrode position can be acquired by using a stylus tracked with a motion capture system.

We propose a solution for a relatively fast acquisition of the position by using the OptiTrack measurement system previously introduced. The tracking, with OptiTrack, is based on the recognition of markers. Through a triangulation, it is possible to measure the position of each electrode independently on the movement of the patient head by using a reference point attached to the patient. The reference point was installed by attaching a rigid marker configuration to a headband placed on the forehead of the patient.

In the adopted EEG configuration, with up to 64 electrodes, it was not possible to install a marker on top of each electrode, since there is no physical space available on the headset. Even adding external markers in key points of the EEG cap is a troublesome operation. An external tool to acquire each electrode position was instead adopted. In detail, a PlayStation VR controller with its OptiTrack markers was used to acquire the target positions. The acquisition can be made by physically positioning one of the controller markers on top of each electrode. When using the tracking cameras, the position of the reference marker and the PlayStation controller is simultaneously acquired, allowing high consistency in the data acquisition even if the patient is moving inside the tracked area. Motive, additionally, provides an error estimation for the position of each marker so that the final measure can be estimated with an uncertainty range. The position that Motive provides is related to the center of the marker sphere. Each sphere has a 12mm diameter, that should be considered when the obtained measurements are used in the analysis of the head model.

An application with a Graphical User Interface was developed to aid in the measurement process. The goal of the application is to provide a simple and intuitive method to obtain the electrodes external position on an EEG headset. For each reading, the estimated error is shown in real-time, allowing a new measure if the accuracy is not considered acceptable. Normal error range are sub-millimeter; higher errors can derive from a poor calibration of the cameras or if the camera field-of-view is partially occluded during the measurement procedure. The system has to be versatile to cover

[1] `https://polhemus.com`

different EEG electrode configuration that may be used to acquire a specific signal. The most common electrodes should be included in the application in order to attach the correct label to each acquired electrode.

NatNetSDK All the data recorded via the OptiTrack tracking system can be accessed through the default Motive application or 2 SDKs: the Camera SDK and the NatNet SDK. The Camera SDK allows direct access to the cameras, for scenarios in which some custom tracking system are deemed necessary. This SDK offers full camera control, from the synchronization to the vector tracking. It is useful when developing custom tracking application, with features that are not offered by the standard Motive application.

To utilize the existing tracking software on a separate system or application, OptiTrack offers the NatNet SDK. This SDK works with a client/server architecture that can be rapidly deployed in a shared network. The client can access the motion capture data streamed by a tracking server. The tracking server can be a custom application or standard tracking software like Motive or ARENA. For this project, Motive was adopted as the tracking server. Motive can send the client several different datasets:

- The Marker set data, containing the marker ID, the position and the orientation in the 3D space and some statistic on the tracking
- A group of markers can be configured as a rigid body. All the spatial information regarding the tracked rigid body can be transmitted, together with the list of the associated markers.
- Several rigid bodies can be grouped in a collection defined as a Skeleton. The Skeleton data contain all the information regarding the elements that compose it, organized hierarchically.
- Other advanced data, such as Force Plate and Analog Devices

With the SDK it is also possible to configure Motive remotely. The SDK provides various APIs for different programming languages like a C++ "Native" client, a .NET managed assembly and also managed client that can be used in other applications such as MATLAB.

5.2 Implementation

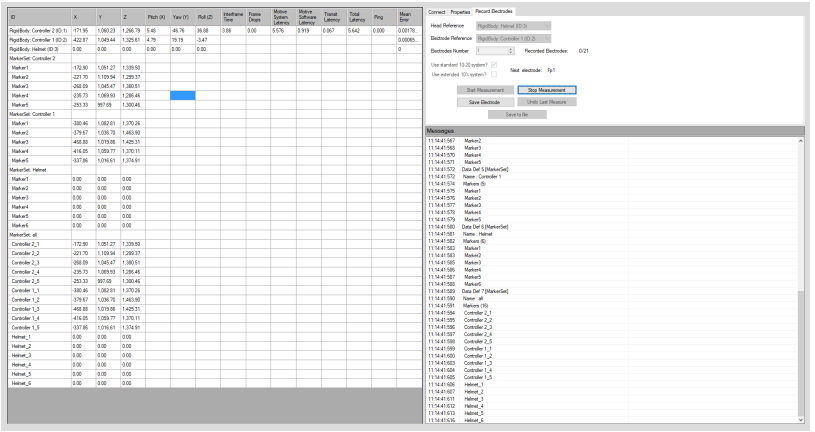


Figure 5.1: NatNet application

The NatNetSDK was implemented using the C# integration and was integrated in a Windows Form application. The application offers several features to the user.

Information grid A grid that shows several information for each object and marker. The information shown is the positional information of the object (X, Y, Z coordinates and pitch, yaw and roll), network information (frame drops, ping, latency) and the mean error on the positional measure. This grid is updated in real-time, allowing the user to rapidly identify if there are problems with the connection or the tracking.

Message report panel This panel shows a log for the application and shows the values recorded after each electrode measure, allowing the user to redo the measure if needed.

Connection panel Interface used to connect the application to the Motive application. Motive can be run on the same machine and be connected using the local-host interface or in another computer connected in the same local area network. This allows us to have a computer dedicated to the EEG recording and electrode position measurement,

while another workstation manages the OptiTrack tracking system. The connection itself can be done using unicast, multicast or broadcast, giving maximum flexibility in the connection setup. To start the connection, the user has to choose the local interface used for the connection and the remote IP for the server (set by default to the local-host 127.0.0.1).

Property panel Used to configure Motive remotely. With this panel, it is possible to enable or disable specific asset in the Motive scene, identified by their name. It is also possible to change e configure specific property of the asset. Usually, the configuration should be done beforehand on the Motive application, but this option allows on-the-fly change if some configuration does not work correctly.

Record panel This is the key element of the application. With this panel, it is possible actually to start the measurement of the electrode position. The first setup that must be done is the choice of a tracked reference for the head and the electrode.

To enable an accurate recording independent on the head movement, it is necessary to attach a set of markers to the user head. In the test done, the most comfortable setup was an elastic headband with a set of reflective markers attached. The headband does not interfere with the EEG helmet and can be easily removed after the recording. Other solution can be adopted, to leave the patient in the most pleasant situation achievable.

The electrode reference is designed to be the OptiTrack controller with its marker setup. However, if other configurations are deemed necessary, other markers can be easily selected as a reference. To obtain the best measure possible, the pivot point of each chosen reference for the electrode must be placed on the spherical marker that will touch the electrodes.

After the references are chosen, the operator can pick which electrode configuration to record. Two pre-configured setup are already provided, if the EEG electrodes are placed according the common 10-10 standard electrode position [33] or the extended 10-20 [18].

When using the standard electrode position, the system suggests the

name of each electrode that should be measured next, according to the standard nomenclature. To use a non-standard electrode position, such as EEG recording done with a high number of electrodes (e.g., 128 electrodes setups), the operator can select an arbitrary number of positioned electrode before starting the measure. However, since this particular configuration does not have a standard nomenclature or position order, it is not possible to provide the user a suggested order to acquire the values. So the responsibility to acquire the data in a specific sequence is left to the user, according to the measure intended goal. Once a measure is acquired, it can be saved using one of the 3 provided common export format: a .csv spreadsheet, a JSON file or an XML file.

To obtain a valid position measure for each electrode, it is necessary to subtract from the positional and rotational value of the electrode, the measure of the reference point. Assuming the reference is connected with the patient head, with this subtraction, the obtained values are independent from the position of the user. Every value is therefore related to the OptiTrack system origin, located at the center of the room at the ground level. The recorded value for each electrode, and the reference is affected by a mean error, provided by the Motive server. To obtain an uncertainty range for each electrode, the measurement error theory has to be applied. Since the error is dependent on each other, the Mean Errors provided by OptiTrack has to be added.

5.3 Results

Overall, the time needed to acquire each electrode position is in the order of few seconds. The measure operation, by itself, is fast, but the identification of the electrode position on the cap can be a lengthy operation, especially if the operator is not familiar with the standard electrodes placement. However, this problem can be mitigated by taking the measure during the setup phase of the cap. In the wet-based EEG used in this project, for every electrode, it is necessary to put a liquid electrolytic gel between the electrode and the scalp. When using the gTec setup, it is possible to obtain the impedance of each electrode to verify if the application was made correctly.

If the electrodes are prepared following the standard enumeration, it is possible to acquire the position of each electrode between each gel application, reducing the overall setup phase time.

Regarding the precision of the system, we have found that it is mostly dependant on the calibration of the OptiTrack cameras. As expected, the mean error in the position of each electrode has the same order of magnitude of the global error detected by Motive during the Calibration phase. The errors has a range that went from 0.2 to 1 mm, while all the position measure wherein the cm order. To adapt the electrode placement model, these values were adequate to adjust the model to the specific EEG reading.

Chapter 6

Conclusion

6.1 Results

The objective of this thesis was the creation of a new Virtual Reality system that integrates advanced EEG source imaging algorithms to display, in real-time, the brain activity recorded from the EEG. By leveraging on modern brain segmentation techniques, we were able to obtain a realistic model of the human head and a valid representation of the white matter fiber tracts. The meshes were obtained from real human MRI and fMRI, with different resolutions. The system was built to support high-resolution geometries while keeping the frame-rate required for a fluid VR experience.

We implemented an ESI pipeline able to solve the inverse problem with a low delay, even when adopting a high-density research EEG setup. We adopted a modular approach, to distribute the workload of the high-density EEG elaboration on multiple machines. The pipeline was tested with both simulated data and real EEG recording. The real-time elaboration was stable even when adopting a high-resolution headset, with 64 or 128 electrodes. By making use of the Unity3D XR library, we were able to integrate the visualization on a modern VR headset, while developing a custom tracking system by making use of an external tracking technology. The application can be used as a standalone desktop experience or in a 3D immersive environment designed to be explored with a VR headset and with the OptiTrack tracking technology.

The system was adopted in several demonstrations, with EEG recording

acquired directly with an EEG cap, or with pre-recorded data. The result of this thesis has shown this project can be easily adapted for Neurofeedback treatments or for research and diagnostic application. With the MRI and fMRI data of a patient, it is possible to reconstruct an accurate model of its brain structure. This information can be exploited to obtain a precise analysis of the brain activity while localizing the activation source through the ESI pipeline.

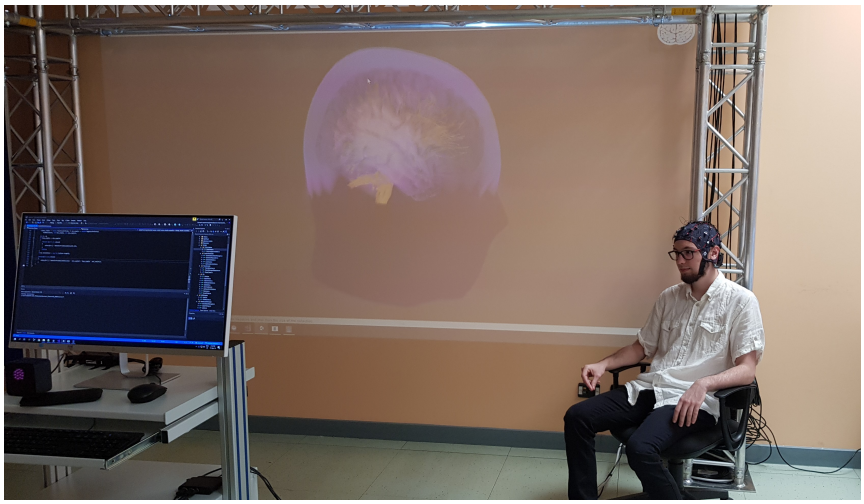


Figure 6.1: Live demonstration of the Neurosurf application and Pipeline. The data are acquired from a 32-electrodes wireless EEG cap, and the VR scene is also projected on the wall

6.2 Future Works

Application Device-Independent All the work that has been done for this thesis has been designed around the available hardware, headset, and tracking system. A major feature that can be introduced in the project is the separation of the software component related to the source imaging pipeline and the mesh visualization and management from the HTC VIVE Headset and the OptiTrack tracking modules. By creating a unified API or an add-on for Unity, it will be possible to integrate the developed technologies on

a large variety of platforms. The mesh import and configuration should be streamlined and, possibly, automatized. The source imaging pipeline, given its modularity, can that be executed on different workstation or even on remote server, allowing the visualization also on lower-end VR headsets and workstations.

Numerical Information To extend the usage of the proposed application in an effective medical environment, more information should be provided to the user. The colormap can give a quantitative indication of the activation of the various area of the brain. However, if a more accurate research needs to be done, the numerical value extracted from the EEG should be provided to the user. This will require the creation of two different User Interfaces for the desktop user and the Virtual-Reality user. Displaying a large volume of text or graphs, it's a common practice when using technical application on a desktop setup, but fonts and 2D-graphs are usually not designed for a VR experience. Text has to be positioned carefully in the environment, to not occlude large parts of the user field of view. Moreover, the text dimension is strongly limited by the device resolution, even top-of-the-line consumer headset like the HTC Vive. Furthermore, it should be possible to identify the specific EEG frequency band during the visualization. These bands are used to study specific brain behavior and should be considered individually for a visualization purpose.

Appendices

Appendix A - Conversion from .VTK to .OBJ

```
import vtk
import os
import sys


def createResultFolderIfMissing (base_folder, new_folder_name):
    joined = os.path.join(base_folder, new_folder_name)
    if not os.path.exists(joined):
        os.makedirs(joined)


def vtkOBJWriter (vtkPolyData, labelName, outputFileName):

    file = open(outputFileName, 'w')

    file.write("# wavefront obj file generated from a .vtk file\n")
    file.write("o "+labelName+"\n")

    print("Converting points ...")
    # writing points v, if any
    for pointIndex in range(vtkPolyData.GetNumberOfPoints()):
        point = vtkPolyData.GetPoint(pointIndex)
        x = str(round(point[0], 6))
        y = str((round(point[1],6)))
        z = str(round(point[2],6))
        file.write("v " + x + " " + y + " " + z + "\n")

    print("converting normals ...")
    # writing normals if any. We are not calculating the normals if they are missing
    vtkNormals = vtkPolyData.GetPointData().GetNormals()
    if vtkNormals:
        for normalIndex in range(vtkNormals.GetNumberOfTuples()):
            normal = vtkNormals.GetTuple(normalIndex)
            file.write("vn " + str(round(normal[0], 6))
                       + " " + str(round(normal[1], 6))
                       + " " + str(round(normal[2], 6)) + "\n")

```

```
38      print("converting texture coordinates ...")
39      # writing texture coordinates if any
40      vtkTCoords = vtkPolyData.GetPointData().GetTCoords()
41      if(vtkTCoords):
42          for tCoordsIndex in range(vtkTCoords.
            GetNumberOfTuples()):
43              tCoord = vtkTCoords.GetTuple(tCoordsIndex)
44              file.write("vn " + str(round(tCoord[0], 6))
45                         + " " + str(round(tCoord[1], 6))
46                         + " " + str(round(tCoord[2], 6)) + "\
                               n")
47
48      # No materials and no smoothing
49      file.write("usemtl None\n")
50      file.write("s off\n")
51
52      print("converting vertexes ...")
53      writing verts
54      vertNum = vtkPolyData.GetNumberOfVerts()
55      verts = vtkPolyData.GetVerts()
56      if vertNum > 0:
57      if verts:
58          verts = vtkPolyData.GetVerts()
59          id_list = vtk.vtkIdList()
60          for i in range(0,verts.len()):
61              file.write("p ")
62               verts.GetNextCell(id_list)
63              for j in range(0,id_list.GetNumberOfIds()):
64                  file.write(str(id_list.GetId(j)+1))
65              file.write("\n")
66
67      print("converting lines ...")
68      # writing lines
69      linesNum = vtkPolyData.GetNumberOfLines()
70      if linesNum > 0:
71          lines = vtkPolyData.GetLines()
72          id_list = vtk.vtkIdList()
73          for i in range(0,linesNum):
74              file.write("l ")
75              lines.GetNextCell(id_list)
76              for j in range(0,id_list.GetNumberOfIds()):
77                  if vtkTCoords:
78                      file.write(str(id_list.GetId(j)+1) + "/"
                               + str(id_list.GetId(j)+1) + " ")
79                  else:
```

```
80                      file.write(str(id_list.GetId(j)+1)+" ")
81              file.write("\n")
82
83      print("converting polys ...")
84      # writing polys
85      polyNum = vtkPolyData.GetNumberOfPolys()
86      if polyNum > 0:
87          polys = vtkPolyData.GetPolys()
88          id_list = vtk.vtkIdList()
89          for i in range(0, polyNum):
90              file.write("f ")
91              polys.GetNextCell(id_list)
92              for j in range(0, id_list.GetNumberOfIds()):
93                  if vtkNormals:
94                      if vtkTCoords:
95                          file.write(str(id_list.GetId(j)+1) +
                                "/" + str(id_list.GetId(j)+1) +
                                "/"+str(id_list.GetId(j)+1)+" "
                               )
96                      else:
97                          file.write(str(id_list.GetId(j)+1) +
                                "//" + str(id_list.GetId(j)+1)
                               + " ")
98                  else:
99                      if vtkTCoords:
100                         file.write(str(id_list.GetId(j)+1) +
                                "/" + str(id_list.GetId(j)+1) +
                                " ")
101                     else:
102                         file.write(str(id_list.GetId(j)+1) +
                                " ")
103             file.write("\n")
104
105     print("converting strips ...")
106     # writing strips
107     tStripsNum = vtkPolyData.GetNumberOfStrips()
108     if tStripsNum > 0:
109         tStrips = vtkPolyData.GetStrips()
110         id_list = vtk.vtkIdList()
111         for i in range(0,tStripsNum):
112             for j in range(2, id_list.GetNumberOfIds()):
113                 j1 = j-1
114                 j2 = j-2
115                 if vtkNormals:
116                     if vtkTCoords:
```

```
117                             output = "f "+str(id_list[j1]+1)+"/"
                                    +str(id_list[j1]+1)+"/"+str(
                                    id_list[j1]+1)+" "
118                             output += str(id_list[j2]+1)+"/"+str
                                    (id_list[j2]+1)+"/"+str(id_list[
                                    j2]+1)+" "
119                             output += str(id_list[j]+1) + "/" +
                                    str(id_list[j]+1) + "/" + str(
                                    id_list[j]+1) + "\n"
120                             file.write(output)
121                         else:
122                             output = "f " + str(id_list[j1]+1) +
                                     "//" + str(id_list[j1]+1) + " "
123                             output += str(id_list[j2]+1) + "//"
                                    + str(id_list[j2]+1) + " "
124                             output += str(id_list[j]+1) + "//" +
                                     str(id_list[j]+1) + "\n"
125                             file.write(output)
126                 else:
127                     if vtkTCoords:
128                         output = "f " + str(id_list[j1]+1) +
                                     "/" + str(id_list[j1]+1) + " "
129                         output += str(id_list[j2]+1) + "/" +
                                     str(id_list[j2]+1) + " "
130                         output += str(id_list[j]+1) + "/" +
                                    str(id_list[j]+1) + "\n"
131                         file.write(output)
132                     else:
133                         output = "f " + str(id_list[j1]+1) +
                                     " "
134                         output += str(id_list[j2]+1) + " "
135                         output += str(id_list[j]+1) + "\n"
136                         file.write(output)
137
138
139 def main(inputFolder):
140     createResultFolderIfMissing(inputFolder, "converted")
141     for file in os.listdir(inputFolder):
142         if file.endswith(".vtk"):
143             file_name = os.path.splitext(file)
144             input_path = os.path.join(inputFolder, file)
145             output_path = os.path.join(inputFolder, "
                    converted", file_name[0] + ".obj")
146
147             print("Reading vtk file: " + file)
```

```
148             reader = vtk.vtkGenericDataObjectReader()
149             reader.SetFileName(input_path)
150             reader.Update()
151 
152             inputPolyData = reader.GetOutput()
153 
154             print("Starting obj conversion")
155             vtkOBJWriter(inputPolyData, file_name[0],
                    output_path)
156 
157 
158 # main program here
159 try:
160     inputFolder = str(sys.argv[1])
161 except:
162     print("Missing or not valid folder argument")
163 
164 if os.path.exists(inputFolder):
165     main(inputFolder)
166 else:
167     print("The input is not a valid folder")
```

Bibliography

[1] *Encyclopedia of Neuroscience | ScienceDirect*, https://www.sciencedirect.com/referencework/9780080450469/encyclopedia-of-neuroscience.

[2] *How to maximize AR and VR performance with advanced stereo rendering – Unity Blog*, https://blogs.unity3d.com/2017/11/21/how-to-maximize-ar-and-vr-performance-with-advanced-stereo-rendering/.

[3] *Observer Design Pattern | Microsoft Docs*, https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern.

[4] *sccn/labstreaminglayer*, September 2019, original-date: 2018-02-28T10:50:12Z.

[5] T. Blum, R. Stauder, E. Euler, and N. Navab, *Superman-like X-ray vision: Towards brain-computer interfaces for medical augmented reality*, 2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), November 2012, pp. 271–272.

[6] D. Borland and R. M. Taylor Ii, *Rainbow Color Map (Still) Considered Harmful*, IEEE Computer Graphics and Applications **27** (2007), no. 2, 14–17.

[7] Borís Burle, Laure Spieser, Clémence Roger, Laurence Casini, Thierry Hasbroucq, and Franck Vidal, *Spatial and temporal resolutions of EEG: Is it really black and white? A scalp current density view*, International Journal of Psychophysiology **97** (2015), no. 3, 210–220.

[8] Baek-Hwan Cho, Saebyul Kim, Dong Ik Shin, Jang Han Lee, Sang Min Lee, In Young Kim, and Sun I. Kim, *Neurofeedback Training with Virtual Reality for Inattention and Impulsiveness*, CyberPsychology & Behavior **7** (2004), no. 5, 519–526.

[9] C. G. Coogan and B. He, *Brain-Computer Interface Control in a Virtual Reality Environment and Applications for the Internet of Things*, IEEE Access **6** (2018), 10840–10849.

[10] Rahul S. Desikan, Florent Ségonne, Bruce Fischl, Brian T. Quinn, Bradford C. Dickerson, Deborah Blacker, Randy L. Buckner, Anders M. Dale, R. Paul Maguire, Bradley T. Hyman, Marilyn S. Albert, and Ronald J. Killiany, *An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest*, NeuroImage **31** (2006), no. 3, 968–980.

[11] Longwei Fang, Lichi Zhang, Dong Nie, Xiaohuan Cao, Islem Rekik, Seong-Whan Lee, Huiguang He, and Dinggang Shen, *Automatic brain labeling via multi-atlas guided fully convolutional networks*, Medical Image Analysis **51** (2019), 157–168.

[12] Karl J. Friston, *Functional and effective connectivity in neuroimaging: A synthesis*, Human Brain Mapping **2** (1994), no. 1-2, 56–78 (en), Citation Key Alias: fristonFunctionalEffectiveConnectivity1994a.

[13] Eleftherios Garyfallidis, Matthew Brett, Marta Morgado Correia, Guy B. Williams, and Ian Nimmo-Smith, *QuickBundles, a Method for Tractography Simplification*, Frontiers in Neuroscience **6** (2012) (English).

[14] Rémi Gillig, *speps/XInputDotNet*, September 2019, https://github.com/speps/XInputDotNet.

[15] Rolando Grave de Peralta Menendez, Micah M. Murray, Christoph M. Michel, Roberto Martuzzi, and Sara L. Gonzalez Andino, *Electrical neuroimaging based on biophysical constraints*, NeuroImage **21** (2004), no. 2, 527–539 (eng).

[16] Roberta Grech, Tracey Cassar, Joseph Muscat, Kenneth P. Camilleri, Simon G. Fabri, Michalis Zervakis, Petros Xanthopoulos, Vangelis Sakkalis, and Bart Vanrumste, *Review on solving the inverse problem in EEG source analysis*, Journal of NeuroEngineering and Rehabilitation **5** (2008), no. 1, 25.

[17] Vijay P.B. Grover, Joshua M. Tognarelli, Mary M.E. Crossey, I. Jane Cox, Simon D. Taylor-Robinson, and Mark J.W. McPhail, *Magnetic Resonance Imaging: Principles and Techniques: Lessons for Clinicians*, Journal of Clinical and Experimental Hepatology **5** (2015), no. 3, 246–255.

[18] Tyler Grummett, Richard Leibbrandt, Trent Lewis, Dylan DeLosAngeles, David Powers, John Willoughby, Kenneth Pope, and Sean Fitzgibbon, *Measurement of neural signals from inexpensive, wireless and dry EEG systems*, Physiological Measurement **36** (2015).

[19] L Haas, *Hans Berger (1873–1941), Richard Caton (1842–1926), and electroencephalography*, Journal of Neurology, Neurosurgery, and Psychiatry **74** (2003), no. 1, 9.

[20] Hans Hallez, Bart Vanrumste, Roberta Grech, Joseph Muscat, Wim De Clercq, Anneleen Vergult, Yves D'Asseler, Kenneth P Camilleri, Simon G Fabri, Sabine Van Huffel, and Ignace Lemahieu, *Review on solving the forward problem in EEG source analysis*, Journal of NeuroEngineering and Rehabilitation **4** (2007), no. 1, 46 (en).

[21] Sofie Therese Hansen, Søren Hauberg, and Lars Kai Hansen, *Data-driven forward model inference for EEG brain imaging*, NeuroImage **139** (2016), 249–258.

[22] Lars G Hanson, *Introduction to Magnetic Resonance Imaging Techniques*, 48 (en).

[23] Lawrence J. Hettinger and Gary E. Riccio, *Visually Induced Motion Sickness in Virtual Environments*, Presence: Teleoperators and Virtual Environments **1** (1992), no. 3, 306–310.

[24] Michael R. Hodge, William Horton, Timothy Brown, Rick Herrick, Timothy Olsen, Michael E. Hileman, Michael McKay, Kevin A. Archie, Eileen Cler, Michael P. Harms, Gregory C. Burgess, Matthew F. Glasser, Jennifer S. Elam, Sandra W. Curtiss, Deanna M. Barch, Robert Oostenveld, Linda J. Larson-Prior, Kamil Ugurbil, David C. Van Essen, and Daniel S. Marcus, *ConnectomeDB—Sharing human brain connectivity data*, NeuroImage **124** (2016), 1102–1107.

[25] Scott A Huettel, Allen W Song, and Gregory McCarthy, *Functional Magnetic Resonance Imaging, Second Edition*, 8 (en).

[26] Ben Jeurissen, Maxime Descoteaux, Susumu Mori, and Alexander Leemans, *Diffusion MRI fiber tractography of the brain*, NMR in biomedicine **32** (2019), no. 4, e3785 (eng).

[27] Derek K. Jones, Thomas R. Knösche, and Robert Turner, *White matter integrity, fiber count, and other fallacies: The do's and don'ts of diffusion MRI*, NeuroImage **73** (2013), 239–254.

[28] Anisha Keshavan, Arno Klein, and Ben Cipollini, *Interactive online brain shape visualization*, Research Ideas and Outcomes **3** (2017),

e12358 (en).

[29] Seung-Wook Kim and Joon-Kyung Seong, *Virtual Display of 3d Computational Human Brain Using Oculus Rift*, Design, User Experience, and Usability: Technological Contexts (Aaron Marcus, ed.), Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 258–265 (en).

[30] Arno Klein, Satrajit S. Ghosh, Forrest S. Bao, Joachim Giard, Yrjö Häme, Eliezer Stavsky, Noah Lee, Brian Rossa, Martin Reuter, Elias Chaibub Neto, and Anisha Keshavan, *Mindboggling morphometry of human brains*, PLOS Computational Biology **13** (2017), no. 2, e1005350 (en).

[31] Arno Klein and Jason Tourville, *101 Labeled Brain Images and a Consistent Human Cortical Labeling Protocol*, Frontiers in Neuroscience **6** (2012) (English).

[32] J. L. Lancaster, L. H. Rainey, J. L. Summerlin, C. S. Freitas, P. T. Fox, A. C. Evans, A. W. Toga, and J. C. Mazziotta, *Automated labeling of the human brain: A preliminary report on the development and evaluation of a forward-transform method*, Human Brain Mapping **5** (1997), no. 4, 238–242 (en).

[33] Jian Le, Min Lu, Emiliana Pellouchoud, and Alan Gevins, *A rapid method for determining standard 10/10 electrode positions for high resolution EEG studies*, Electroencephalography and Clinical Neurophysiology **106** (1998), no. 6, 554–558.

[34] A. Lécuyer, F. Lotte, R. B. Reilly, R. Leeb, M. Hirose, and M. Slater, *Brain-Computer Interfaces, Virtual Reality, and Videogames*, Computer **41** (2008), no. 10, 66–72, Citation Key Alias: lecuyerBrainComputerInterfacesVirtual2008a.

[35] Hengameh Marzbani, Hamid Reza Marateb, and Marjan Mansourian, *Neurofeedback: A Comprehensive Review on System Design, Methodology and Clinical Applications*, Basic and Clinical Neuroscience **7** (2016), no. 2, 143–158.

[36] Christoph M. Michel and Denis Brunet, *EEG Source Imaging: A Practical Review of the Analysis Steps*, Frontiers in Neurology **10** (2019), 325 (eng).

[37] Christoph M. Michel, Micah M. Murray, Göran Lantz, Sara Gonzalez, Laurent Spinelli, and Rolando Grave de Peralta, *EEG source imaging*, Clinical Neurophysiology **115** (2004), no. 10, 2195–2222 (en).

[38] Kenneth Moreland, *Color Map Advice for Scientific Visualization*, https://www.kennethmoreland.com/color-advice/.

[39] Kenneth Moreland, *Why We Use Bad Color Maps and What You Can Do About It*, Electronic Imaging **2016** (2016), no. 16, 1–6 (en).

[40] Tim Mullen, Christian Kothe, Yu Mike Chi, Alejandro Ojeda, Trevor Kerth, Scott Makeig, Gert Cauwenberghs, and Tzyy-Ping Jung, *Real-time modeling and 3d visualization of source dynamics and connectivity using wearable EEG*, Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference **2013** (2013), 2184–2187 (eng).

[41] S. Ogawa, T. M. Lee, A. R. Kay, and D. W. Tank, *Brain magnetic resonance imaging with contrast dependent on blood oxygenation*, Proceedings of the National Academy of Sciences **87** (1990), no. 24, 9868–9872 (en).

[42] R D Pascual-Marqui, *Standardized low resolution brain electromagnetic*, Clinical Pharmacology (2002), 16 (en).

[43] Roberto D. Pascual-Marqui, Michaela Esslen, Kieko Kochi, and Dietrich Lehmann, *Functional imaging with low-resolution brain electromagnetic tomography (LORETA): a review.*, Methods and findings in experimental and clinical pharmacology **24** (2002), no. Suppl, 91–95.

[44] F. Perrin, J. Pernier, O. Bertrand, and J. F. Echallier, *Spherical splines for scalp potential and current density mapping*, Electroencephalography and Clinical Neurophysiology **72** (1989), no. 2, 184–187, Citation Key Alias: perrinSphericalSplinesScalp1989a.

[45] François Tadel, Sylvain Baillet, John C. Mosher, Dimitrios Pantazis, and Richard M. Leahy, *Brainstorm: A User-Friendly Application for MEG/EEG Analysis*, 2011.

[46] Richard F Thompson, *The brain : an introduction to neuroscience / Richard F. Thompson*, The brain an introduction to neuroscience, A series of books in psychology, Freeman, New York, 1985 (English).

[47] Mullen Tim, *The Glass Brain - Tim Mullen | Neuroscience*, http://www.antillipsi.net/art-1/the-glass-brain.

[48] Andries van Dam, David H Laidlaw, and Rosemary Michelle Simpson, *Experiments in Immersive Virtual Reality for Scientific Visualization*, Computers & Graphics **26** (2002), no. 4, 535–555.

[49] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux, *The*

NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering **13** (2011), no. 2, 22–30.

[50] Gavin Wheeler, Shujie Deng, Nicolas Toussaint, Kuberan Pushparajah, Julia Schnabel, John Simpson, and Alberto Gomez, *Virtual Interaction and Visualisation of 3d Medical Imaging Data with VTK and Unity*, Healthcare Technology Letters **5** (2018).

[51] Clinton Johan Ade Wicaksono, Susy Suswaty, Nursama Heru Apriantoro, and Ary Sasongko, *Image Quality Analysis 4 Chamber Sections of Cardiac MRI with and without utilizing shim volume in the steady state free precession sequences*, Journal of Vocational Health Studies **1** (2018), no. 3, 97–101 (id-ID).

[52] QiBin Zhao, LiQing Zhang, and Andrzej Cichocki, *EEG-based asynchronous BCI control of a car in 3d virtual reality environments*, Chinese Science Bulletin **54** (2009), no. 1, 78–87 (en).

Acronyms

API	Application Program Interface.
AR	Augmented Reality.
BCI	Brain-Computer Interfaces.
BEM	Boundary Element Method.
BOLD	Blood Oxygenation Level.
CPU	central processing unity.
CSF	Cerebrospinal fluid.
CSV	Comma-separated values.
CT	computed tomography.
DIS	Distributed Inverse Solution.
DKT	Desikan-Killiany-Tourville.
dMRI	diffusion magnetic resonance imaging.
DoF	Degree of Freedom.
DOTS	Data-OrientedTechnology Stack.
DTI	Diffusion Tensor Imaging.
EEG	Electroencephalography.
EPSP	excitatory postsynaptic potential.
ESI	EEG Source Imaging.
FEM	Finite Element Method.
fMRI	Functional MRI.
FP	forward problem.

GPU	graphics processing unit.
HDPR	High-Definition Render Pipeline.
iEEG	intracranial EEG.
IL2CPP	Intermediate Language To C++.
IP	inverse problem.
IPSP	inhibitory postsynaptic potential.
JSON	JavaScript Object Notation.
LSL	Lab Streaming Layer.
MEG	Magnetoencephalography.
MN	Minimum Norm.
MR	Mixed Reality.
MRI	Magnetic Resonance Imaging.
NIRS	Near-infrared spectroscopy.
NTP	Network Time Protocol.
PET	Positron Emission Tomography.
RAM	Random Access Memory.
SDK	Software Development Kit.
UI	User Interface.
VR	Virtual Reality.
VRPN	Virtual-Reality Peripheral Network.
VTK	Visualization Toolkit.
WMN	Weighted Minimum Norm.
XML	eXtensible Markup Language.