UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL DE FI DE GRAU

**Grau en Enginyeria Biomèdica**

# DEVELOPMENT OF A WEB-BASED GRAPHICAL USER INTERFACE TO DESIGN BRAIN FIBER MODELS FOR TRACTOGRAPHY VALIDATION

## Volum I

## Memòria – Pressupost


**Autor:** Guillem González Vela
**Director:** Jordi Solà Soler
**Departament** ESAII
**Co-Director**: Emmanuel Caruyer

**Convocatòria:** Juny de 2017

# Contents

# List of Figures

# List of Tables

# Abstract

Diffusion Magnetic Resonance Imaging (MRI) is an advanced MRI technique which can provide brain white matter tissue microscopic information. From this information, the connectivity map of axons in the brain can be obtained using tractography algorithms. However, this cartography of the brain wiring is known to suffer from several biases.

*Phantomαs* is an open source library created with the aim of evaluating tractography. It allows the creation of *in silico* brain phantoms and simulates its diffusion weighted MR images. Tractograms obtained from these MR images can be compared to the ground truth. The trajectories of the tracts are provided to *Phantomαs* in a hand written plain text file. This process can be time consuming and tedious for the users.

The aim of this project is to create a graphical user interface (GUI) for designing phantoms and generating corresponding description files for *Phantomαs*. We developed a software that runs in a web-based user interface. It enables users to interact with a 3D representation of a phantom and manually edit any property to generate the corresponding description file.

# Resum

La Imatgeria per Ressonància Magnètica (MRI per les seves sigles en anglès) de difusió és una tècnica avançada d'MRI que pot proporcionar informació sobre el teixit microscòpic de la matèria blanca del cervell. A partir d'aquesta informació i mitjançant algorismes de tractografia, pot obtenir-se el mapa de connexions entre axons al cervell. És sabut que aquesta cartografia de connexions del cervell sol presentar nombrosos biaixos.

*Phantomαs* és una llibreria de codi lliure creada amb l'objectiu d'avaluar algorismes de tractografia. Permet crear fantomes *in silico* i simula les seves imatges potenciades en difusió. Les tractografies obtingudes a partir d'aquestes imatges poden ser comparades amb el model inicial. Les trajectòries dels tractes es proporcionen a *Phantomαs* mitjançant fitxers de text pla escrits a mà. Aquest procés pot comportar molt de temps i resultar feixuc als usuaris.

L'objectiu d'aquest projecte és crear una interfície gràfica d'usuari (GUI per les seves sigles en anglès) per dissenyar fantomes i generar els seus corresponents fitxers de descripció per *Phantomαs*. El programari desenvolupat s'executa en una interfície d'usuari integrada en un entorn web i permet els usuaris d'interactuar amb una representació 3D del fantoma i editar-ne qualsevol característica per generar el fitxer de descripció.

# Resumen

La Imagería por Resonancia Magnética (MRI por sus siglas en inglés) de difusión es una avanzada técnica de MRI que puede proporcionar información sobre el tejido microscópico de la materia blanca del cerebro. A partir de esta información, mediante algorismos de tractografía puede obtenerse el mapa de conexiones entre axones en el cerebro. Es conocido que esta cartografía de conexiones del cerebro suele presentar numerosos sesgos.

*Phantomαs* es una librería de código libre para evaluar algorismos de tractografía. Permite crear fantomas *in silico* y simula sus imágenes potenciadas en difusión. Las tractografías obtenidas a partir de estas imágenes pueden entonces compararse con el modelo inicial. Las trayectorias de los tractos se proporcionan a *Phantomαs* mediante ficheros de texto llano escritos a mano. Este proceso puede llevar mucho tiempo y resultar pesado para los usuarios.

El objetivo de este proyecto es crear una interficie gráfica de usuario (GUI) para diseñar fantomas y generar sus correspondientes ficheros de descripción para *Phantomαs*. El programa desarrollado es ejecutado en una interficie de usuario integrada en un entorno web y permite a los usuarios interactuar con una representación 3D del fantoma y editar cualquier característica para generar el fichero de descripción.

# Chapter 1

# Introduction

## 1.1 Context

This thesis was developed during an internship carried out during spring 2017 in *Inria Rennes — Bretagne Atlantique* research center, hosted by *VisAGeS*, team belonging to both *Inria* and *IRISA* institutes.

### 1.1.1 Center

The research center *Inria Rennes — Bretagne Atlantique* was created in 1980 and concerns two different French research institutes, *Inria* and *IRISA*. It is located in Rennes, in Campus Beaulieu, and is associated with Université de Rennes I.

*Inria* is a French national institute for research in math and informatics. Its acronym stands for "Institut National de Recherche en Informatique et en Automatique", "National Institute of Research in Informatics and Automatics" in English. *Inria* was created in 1967 and it currently employs 2600 people distributed in 9 different locations around France [7].


Figure 1.1: Graphic identity of *Inria* institute

IRISA is a mixt research unit for informatics, signal and image treatment and robotics. Its acronym stands for "Institut de recherche en informatique et systèmes aléatoires", "Research institute in computer science and random systems" in English. IRISA was created in 1975 and it currently employs 800 people divided in 40 teams around Brittany [8].

### 1.1.2 Research Team

The research team *VisAGeS (Vision, Action and information manaGement System in health)* is jointly awarded by *INSERM (National Institute of Health and Medical Research)* and *Inria*, belongs to *IRISA* and is located in Rennes, France. It is devoted to the development of new processing algorithms in the context of medical image computing and computer assisted interventions [9].

### 1.1.3 The internship

The internship carried out in *VisAGeS* team was supervised by researcher Emmanuel Caruyer [10], taking place at research center *Inria Rennes — Bretagne Atlantique* from February to June 2017.

Among the research topics of the team, the context of this internship covers the neuroimaging. The main goal is to develop a software application to assist processing algorithms for this kind of applications.

## 1.2 Background

In the last years medical imaging has become one of the most important technologies in medicine, given its capacity of providing a good basis for diagnostics in a non-invasive manner. Any technique concerning medical image strongly lies in hardware and software improvements, while requiring precision at a tiny error margin. This has led medical institutions and companies to invest in research for developing more and more precise medical imaging platforms.

The project defined in this thesis is based on the medical imaging approach for the neural tracts. This information is taken from diffusion-weighted images captured in a magnetic resonance procedure.

### 1.2.1 Diffusion MRI

Diffusion-weighted Magnetic Resonance Imaging (DWI), commonly known as just "diffusion MRI" is an imaging method used in medicine that generates contrast in magnetic resonance images by using the diffusion of water molecules contained in the subject [11].

The diffusion-weighted signal can be modeled using a tensor, which is the basis of the popular technique known as Diffusion Tensor Imaging (DTI). DTI is commonly used to generate tractographies of white matter in the brain. In this kind of MRI, the diffusion is characterized for each direction of space and the information inscribed in each voxel [12].

### 1.2.2 Tractography

From data collected in Diffusion Weighted Images (DWI) taken in a brain scan, pathways present in brain's white matter can be traced following the principal diffusion directions locally. The computational reconstruction method is known as tractography. Its aim is to describe *in vivo* and precisely the paths present in brain's white matter. This paths are "fibers" and the model itself is the "phantom".

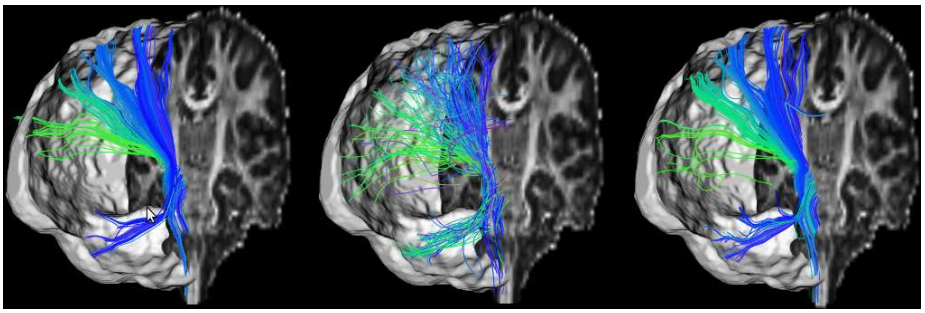


Figure 1.2: Representation of three tractograms of the same subject as interpreted by 3 different tractography algorithms. Courtesy of Emmanuel Caruyer [1].

The tractography has a big computational cost and may be dominated by

false-positive connections [13]. Numerous algorithms have been developed and proposed as new approaches for this kind of analysis to avoid false positives and to improve reliability.

During the development of the reconstruction algorithms, the main problem faced is the validation step. The complex structure of the nervous system demands accuracy in its reconstructions, specially when used for diagnosis. As the target of interest are *in vivo* tissues, the main limitation to validation is that we have no access to ground truth.

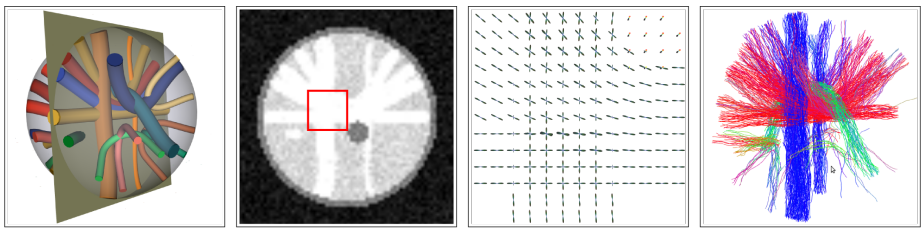
#### Phantom simulation

One of the approaches that allow validation is designing *in silico* phantoms. From the knowledge in MRI technique, the DWI of an eventual scan of these phantoms can be simulated by a software. From the DWI image the algorithms can be tested while exactly knowing the structure of the initial design, and thus, allowing their evaluation.

### 1.2.3 *Phantomαs*

*Phantomαs* [14] is an open-source software developed in *Python* [15] and *C*. It creates realistic phantoms in diffusion MRI, exporting its result to later be reconstructed, allowing the validation of the fiber tracking procedure. This process is illustrated in figure 1.3.

An early version of *Phantomαs* was used to create the testing and training data of the 2nd HARDI Reconstruction Challenge [16], organized at ISBI 2013 [2].



a b c d

Figure 1.3: Illustration of different steps taken by *Phantomαs* [2]. a. An example of fiber bundle configuration, and spherical regions filled with free water. b. Corresponding slice of T1-weighted image. c. Ground truth fiber orientation distribution (zoom of the red square in 1.b). d. Fiber reconstructed with probabilistic streamline tracking. Courtesy of Emmanuel Caruyer.

This project will put all of its attention in the way *Phantomαs* interprets the phantoms and how the users enter their models in.

### Phantoms in *Phantomαs*

In *Phantomαs*, phantoms are to be contained in a spherical cortical area. Fiber bundles are defined by a series of control points. These points are linked by the trajectory of the fiber, computed by the software. The radius of the fiber is also user-specified, being constant through all the path. Consulting the *Phantomαs*' documentation [3] and its source code [4] we can understand the way the trajectory is computed.

Between each pair of points, a 3rd-order polynomial is defined as the path of the fiber. In order to define a 3rd-order polynomial, four constraints are needed. Those are both points' position in the 3D space and the tangent of the trajectory in each of them. An example in 2D may be seen in figure 1.4.

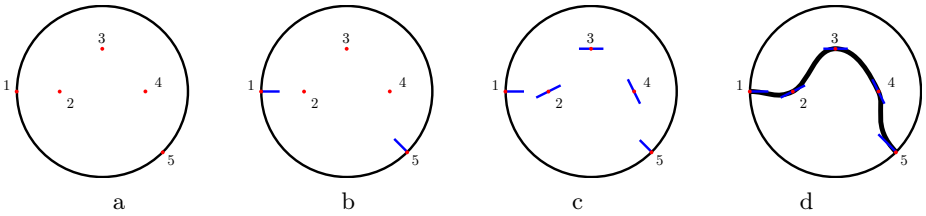



Figure 1.4: Process of computing the trajectory for a fiber bundle in `"symmetric"` tangent mode [3]. a. The given control points. b. First and last derivatives definition. c. Middle tangents calculus. d. Path interpolation.

First and last points get assigned as their derivative the perpendicular direction to the surface of the spherical cortical area. The way *Phantomαs* computes other points' tangents is the only parameter the user is allowed to modify in this process.

There are three different options available for calculating these tangents:

1. **`symmetric`**: Tangents take the resultant slope of the the straight linking the point before and the point after.
2. **`incoming`**: Tangents take the resultant slope of the the straight linking the point in question and the point before.
3. **`outgoing`**: Tangents take the resultant slope of the the straight linking the point in question and the point after.

By default, `symmetric` tangents is set; it is also the one featured in figure 1.4 example.

Apart from defining fibers, *Phantomαs* also allows to define "isotropic regions", which represent cavities in the brain filled with fluid. At the moment only spherical isotropic regions are supported.

Phantom models in are defined in a JSON ("JavaScript Object Notation") format, usually saved as plain text files. *Phantomαs* includes several phantom examples. The library also includes a script for displaying the model a three-dimensional interactive way. This script is `phantomas_view`.

A JSON file contains data in a human-readable structure, which makes phantom design easier. For defining a phantom, *Phantomαs* parses the file, which must contain an object with all the fibers and another with all the isotropic regions contained. One of them is needed so a phantom is interpreted, but not both required at the same time. The elements and their characteristics must be included in the JSON file with the structure defined in figure 1.5.

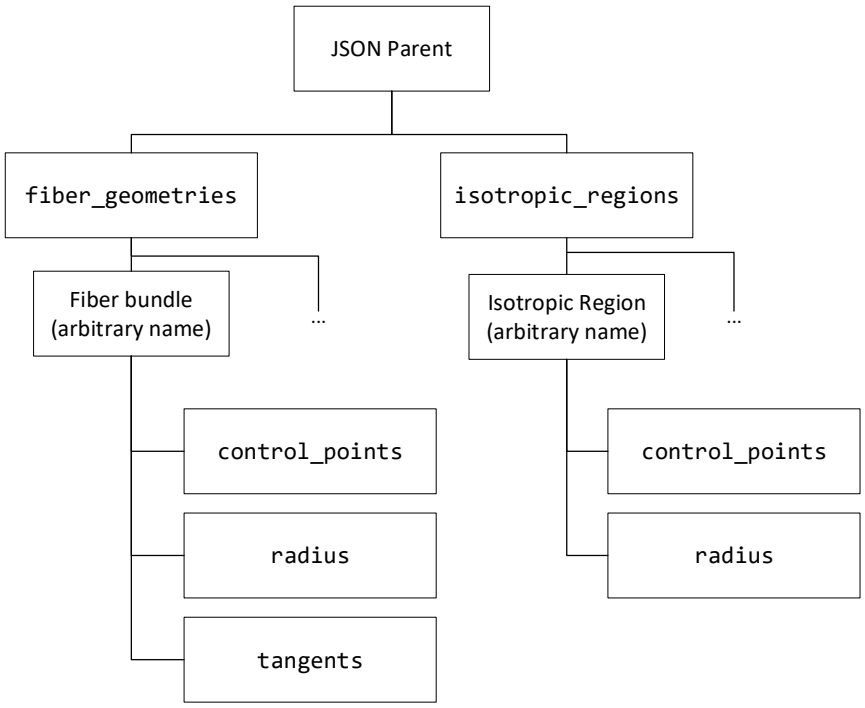



Figure 1.5: Structure of a JSON phantom description for *Phantomαs*. As many as `fiber_geometries` and `isotropic_regions` children needed may be added.

An example of a simple phantom described in a JSON file and its representation can be consulted in figure 1.6. This is the simplest example contained in *Phantomαs* [4]. Note that fibers' control points are expressed in a single array with the structure shown in figure 1.7 and that the fiber bundle present in figure 1.4 is the same as described in figure 1.6.

```
"fiber_geometries" : {
  "fiber_name": {
    "control_points":
      [-10.0 ,  0.0 , 0.0,
        -5.0 ,  0.0 , 0.0,
         0.0 ,  5.0 , 0.0,
         5.0 ,  0.0 , 0.0,
         7.07, -7.07, 0.0],
    "tangents": "symmetric",
    "radius": 2.0
  }
},
"isotropic_regions": {
  "region_number": {
    "radius" : 3.0,
    "center": [0.0, 0.0, 0.0]
  }
}
```

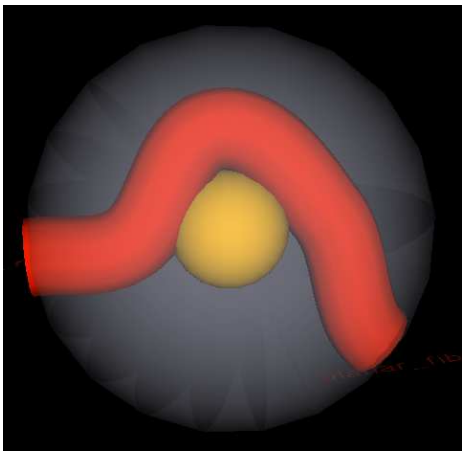


Figure 1.6: An example of a phantom description contained in a JSON file for *Phantomαs* [4] and the representation of the phantom described as rendered by `phantomas_view` script [1]. Its points and trajectory representation process are exemplified with this same fiber bundle in figure 1.4.

```
"control_points":
  [ x_1 ,  y_1 , z_1,
    x_2 ,  y_2 , z_2,
    ... , ... , ... ,
    x_n ,  y_n , z_n ]
```

Figure 1.7: Control points must be contained in a single array with as many as coordinates present in the fiber bundle description.

## 1.3 Objective

The current version of *Phantomαs* is used via the command-line and does require a sometimes complex installation process, making it not user-friendly. The long-term objective is to create a software as a service platform that allows users to define their own phantoms, to set the simulation and to download its result.

The objective of this project is to contribute in the development of this web platform by developing "*Phantomαs* Web Designer", a web-based interactive tool for the creation, edition, visualization and analysis of phantoms. This application must allow users to modify and/or create their own JSON files for *Phantomαs* without having to edit the file itself, while having the knowledge at all time on how their designed phantom will look like. An user-friendly graphical interface and its compatibility with the different devices the users might use must be carefully taken into account. In addition, this tool must be entirely compatible with *Phantomαs*.

# Chapter 2

# Methodology

## 2.1 Requirements

To accomplish its objective, the resulting application must feature certain functionalities to allow the user to edit any aspect of a phantom without leaving the interface. These basic requirements of the application and its interface must carefully be taken into account during its design.

The requirements can be summarized in the following five points:

**a) *Phantomαs* cross-compatibility**
In *Phantomαs*, phantom data is contained in a plain-text JSON file. The user has to be able to load the same file in both *Phantomαs* and the application and receive the same response without any need of modification. The output file should be cross-compatible as well, offering the possibility of either processing it in *Phantomαs* or loading it again for further edition.

**b) Phantom display**
The web canvas must continuously display the phantom model, being the main interaction area for the user. The display should be similar to `phantomas_view` package (see figure 1.6) included in *Phantomαs* to prevent any confusion and to make the cross-compatibility of both tools clear. Interaction with the model should be maximized and navigation through different elements eased, enabling it to be used as a phantom-analysis tool as well.

**c) WYSIWYG interactive edition**
The edition has to be interactive and intuitive. A *what-you-see-is-what-you-get* manner should be implemented, so the user sees at all time how modifications are taking place. Each element's characteristics must be well defined and consequences of the actions taken should be expected by the user. The scene must be focused at all time on the element being currently edited. Guidance edition tools are also desired to make the edition task easier.

**d) Wide edition control**
The edition of any relevant property of the phantom regarding its later processing must be considered, as well as the creation of elements and the creation of phantoms from scratch. These involve:

- Creating and removing fibers and isotropic regions
- Editing general fibers' and isotropic regions' properties, such as radius or tangent-computing method.
- Creating and removing fibers' control points
- Moving fibers' control points and isotropic regions' center point

**e) Versatility**
The application must be designed to be executed in any computer by any user. Cross-platform, low hardware requirements, user-level set up and least dependencies are important points to be taken into account. Only desktop environments are expected to run this application; no mobile device is considered.

## 2.2 Technologies involved

Out of all possibilities available in the scene, a choice had to be made paying special attention to the requirements introduced before. Having the opportunity of building an application from scratch also offers a wide fork of choices.

The fact that *Phantomαs* was coded in *Python* and *C* does not affect the development of the web application, as direct interaction between both is not needed.

The selected language was *JavaScript*. It can be executed in any web navigator, a tool which is widely available. Although it might be also executed in mobile devices, the design is only to be desktop-ready. To run it on internet navigators, *HTML* and *CSS* were employed to create a full web environment. For the 3D graphic representation the *JavaScript* library *THREE.js* was used.

Once the main environment was defined, the used tools may be chosen. Most of these are standard technologies used in web development or were selected by the compatibility with the operative system used; the *GNU/Linux* distribution *Fedora 21* [17]. Each of these technologies used is introduced below.

#### *JavaScript*

*JavaScript* is a widely-implemented programming language [18]. It is part of the core web content production, being employed in every modern web site. All browsers support this language, making it executable for any device able to navigate through the internet. *JavaScript* code is generally nested in an *HTML* page.

*JavaScript* is an object-oriented, functional and interpreted language. This last characteristic makes it act different depending on the interpreter the user is using, and during the design this must be taken into account.

#### *Cascading Style Sheets (CSS)*

*Cascading Style Sheets (CSS)* is the style sheet language used for describing the presentation of *HTML* pages. *CSS* provides many exclusive features and a way to take control of the style of a web page separately from its content.

#### *Hypertext Markup Language (HTML)*

*Hypertext Markup Language (HTML)* is the main language for creating web pages. It is a *Markup* language (not a programming language) and depends on *JavaScript* for executing processes.

An *HTML* document usually contains references to files containing *CSS* code and *JavaScript* code which act on the page. This code can also be nested in the same document as shown in figure 2.1. These three languages conform the major used basis in web development.

The current version of *HTML* standard is *HTML5*, published in October 2014 [19].

```
<html>
<head>
  <title>Hello, World!</title>
  <style>
    body {
      background-color: LightGreen;
      font-family: times;
    }
    input[type=button]:hover {
      background-color: red;
      color: white;
      font-weight: bold;
    }
  </style>
</head>

<body>
  <input type="text" id="nameinput"/>
  <input type="button" value="Set name!" onclick="sayHi()"/>
  <ol id="visitors">
</body>

<script type="text/javascript">
function sayHi() {
  var name = document.getElementById("nameinput").value;
  var visitorList = document.getElementById("visitors");
  var newVisitor = document.createElement("LI");
  newVisitor.innerHTML = name.toString();
  visitorList.appendChild(newVisitor);
}
</script>
</html>
```

Figure 2.1: *HTML* document example, with *CSS* embedded in `<style>` mark and *JavaScript* in `<script>`.

### *THREE.js*

*THREE.js* [20] is a *JavaScript* library for creating animated 3D computer graphics. *THREE.js* supports *WebGL*, an API for rendering graphics in any compatible web browser. As *THREE.js* uses *WebGL* and *JavaScript*, it is cross-browser compatible and does not require any extra plugin. [1]

As *THREE.js* will be responsible for the scene, it will remain active at all time. It is going to be the only *JavaScript* library in the application.

### *Mozilla Firefox*

In addition to interpreting *HTML*, *CSS* and *JavaScript*, desktop web browsers usually come with developer features such as a *JavaScript* console and debugger or a style editor. As the code is not compiled at any time, it might be interpreted in a different way by different browsers.

Setting the functionality of the application as a main priority and not spending much time on solving slight incompatibilities, the application in this project was entirely tested on *Mozilla Firefox* [22] and its development tools. This does not mean it does not work in other web browsers, but that its best performance is found in *Firefox*.

### *Normalize.css*

The present differences between language interpreters might make the web pages look and act in a different way across different browsers: a slight difference in the interpretation of the style may compromise the entire user experience.

*Normalize.css* [23] solves many of this issues by adding specific code for each of the most commonly used browsers so that the subsequent styles act in an homogeneous way. In this project its version 5.0.0 was implemented and tweaked for particular behaviors.

### *Node.js*

*Node.js* [24] is an open-source *JavaScript* environment that allows *JavaScript* code to be run outside the web browser. This allows an easy interaction with the present *JavaScript* environment in a dynamic and lightweight way, making it a widely-used tool in web development. Although its main use is to be run server-side when developing real-time applications, its packages may also be run locally for other purposes.

*Node.js* works using a collection of "modules", and it comes with its own package manager, *npm*. For this project, the modules *http-server* (for testing purposes) and *JSDoc3* were used.

[1] All common desktop browsers do support *WebGL*, although in *Mozilla Firefox* some graphics drivers may be blocked for ensuring stability [21].

#### *JSDoc3*

*JSDoc3* [25] is an open-source API documentation generator for *JavaScript*. Given the code, it parses those comments specially tagged for *JSDoc3*. Out of those comments, the tool generates several structured web documents linked between them, arranging all the elements in the code and easing the navigation through its documentation. Templates and other parameters may be specified for satisfying specific needs. *JSDoc3* is usually used as a *Node.js* module.

#### *reStructuredText*

*reStructuredText* [26] is a lightweight markup language that renders in *HTML*. It is easily readable and its structure and syntax really intuitive, which makes its learning really easy and quick. *reStructuredText* is written in *Python* and usually employed for building documentation of software written in this language.

In this project *reStructuredText* is used for building the user documentation in HTML.

#### *Git* and *GitHub*

*Git* [27] is an open source version control system for tracking changes in collective development of software projects. It allows change uploads to a remote machine, although an online connection is not needed for commiting changes; its information is stored in a specific hidden folder in the same project path.

*GitHub* [28] is a hosting service for *Git* repositories, providing many *Git* functionalities by itself as well as adding its own features. *GitHub* repositories are usually public, although private repositories are available in paid plans.

Both *Git* and *GitHub* were used for the project version control and collaborative source code sharing.

#### *Atom* text editor

*Atom* [29] is an open source text editor developed by *GitHub*. Its code is based os web technologies and offers a large plug-in platform for extensions, usually built using *Node.js*. It also reads the *Git* information and displays changes, branches and commits in the same text editor.

The code for both this project and the thesis itself were written using *Atom*.

## 2.3 Target User

When designing the application it is important to take into account its target user. This involves not only the system configuration, but also a tech-ease analysis for the design.

The users expected to use this application are those who currently build or edit phantom descriptions out of plain text files. This usually concerns researchers and software engineers for tractography computing algorithms, who simulate phantoms in MRI using *Phantomαs*.

*Grosso modo*, the main characteristics of our target users are:

- Advanced knowledge in computer science, a computer is their main tool. Ease in software installation and in internet navigation, having used several kinds of user interfaces.
- Familiar with what a phantom is, how its description file for *Phantomαs* is structured and how it is meant to be used.
- Used to look for and consult user documentation when using software.

From these characteristics we can conclude that the application will not need many instructions as its features are yet expected by the users. The way a phantom or the fiber bundles are modeled does not need to be explained, as users are familiarized with these descriptions.

Regarding the user interface, it has to be as intuitive as possible. Preferably, the interface should be so simple that an usage tutorial is not needed, although user documentation explaining in detail its operation must be made available.

## 2.4 Design

The interactive web page is divided in two kinds of elements. First, the General User Interface (GUI), based in plain *HTML* elements modified on-the-go and which let the user interact with the core of the code by using predefined events. The other part relates to displaying the phantom, based on a *THREE.js* environment using the *WebGL* engine. This structure is schematically represented in figure 2.2. The whole application is hosted in a single *HTML* file that wraps all these elements.

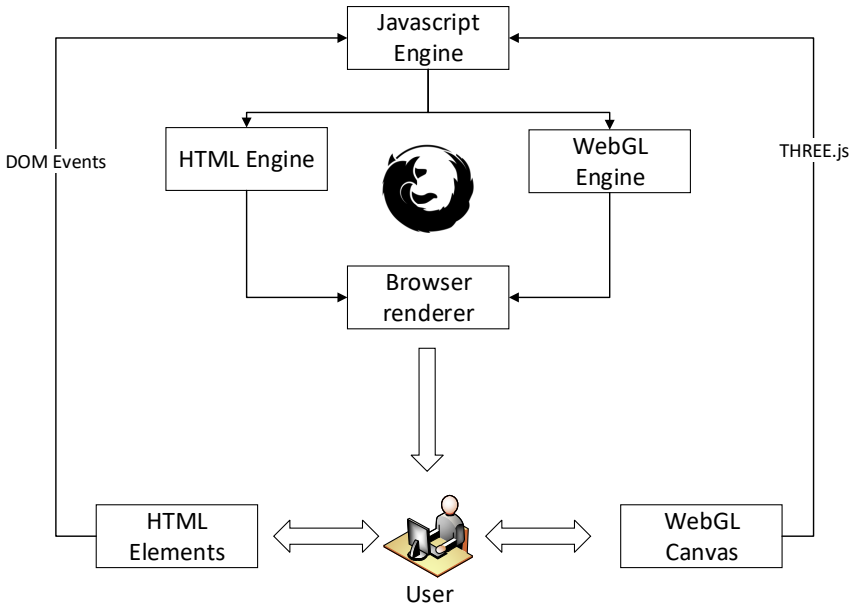



Figure 2.2: Application and browser interaction with the user. Note that the User may interact with two different parts of the app: the *HTML* elements and the *WebGL* canvas.

As the application is expected to be part of a server-side environment, *Phantomαs*' JSON files for phantom description were only set to be loaded from the server and not from the client. As a temporary solution, the path to the files is to be specified as a variable in the URL.

A download prompt was used for downloading the indented `.json` plain-text file containing the generated description of the phantom.

### 2.4.1 User interface

While using the application, the user will be most of the time paying attention to the previsualization canvas, the phantom display area. This is the feature in which the application is mostly based on.

Following this pattern, the user interface design is based on static areas. The main element is the *THREE.js* scene, which features the editing phantom. It is surrounded by HTML elements that guide the user through the edition. Figure 2.3 shows the general user interface appearance.

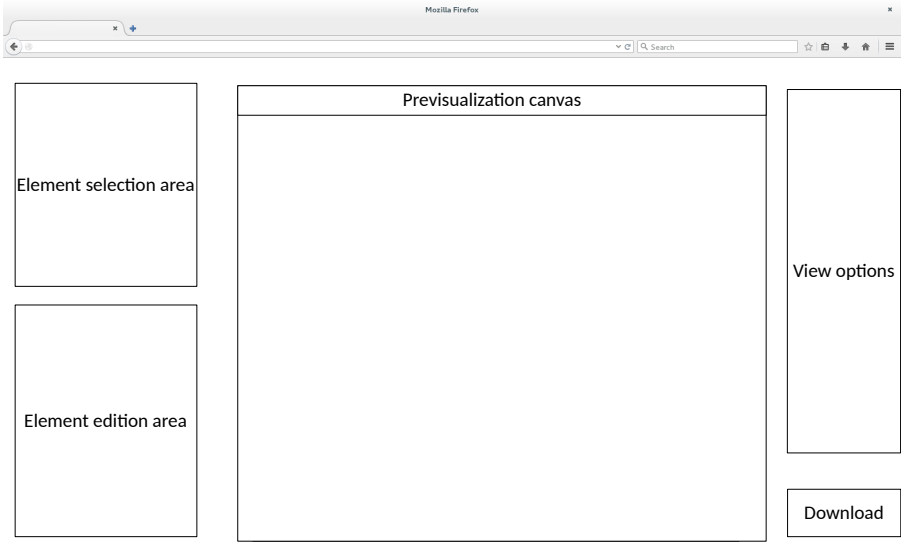



Figure 2.3: User interface schema as shown in the *HTML* page

The scene continuously displays the phantom and allows mouse *drag-and-drop* gestures for rotating and panning, as the wheel is used for zooming in and out. In the view options area (identified in figure 2.3), buttons allow the user to toggle between different displays or positions.

Elements of the phantom (such as fiber bundles and isotropic regions) are continuously shown as a list in the left side of the page, classified by type and marked with their own color in the scene.

For ease of edition, phantom display is tweaked once the user focuses in an element. This involves highlighting it while fading the others and even showing its internal structure while it is being edited. These options may be either enabled or disabled. Fading level may also be selected.

Once an element is selected for edition, its options appear in the element edition area as shown in figure 2.3. This is the only dynamic area and it does not show up unless an element is being edited.

Any edition action immediately takes place in the scene and does not need any saving process. This is part of the *WYSIWYG* design. Control points edition allows an undo action that restores the former position of the point.

Visualization tools are featured at the right side of the page, as shown in figure 2.3. These tools allow the user to change the camera position or tweak the way the phantom is displayed. Their aim is to significantly improve user

experience and usage comfort.

Placed at the right bottom corner, the download button lets the user retrieve the phantom description file at any moment. This does not affect the behavior of the application, whatever the status the user is in.

In order to avoid window scroll-bars, elements in the selection area are automatically re-sized in case the edition elements do not fit in the window. When the screen shape is changed by the user, all the elements in the page automatically re-sized are as well.

### 2.4.2 Phantom display

This is the main part of the application. The phantom display canvas gathers the most part of the page. It is completely interactive and resposible for the *WYSIWYG* user experience (see page 10). Based on *THREE.js*, it is rendered using *WebGL* in an *HTML5* environment (see page 11).

To accomplish the requirement of loading any kind of phantom and displaying it in the canvas by only having its descriptor, we created many classes[2] connected to each other to manage and simplify this functionality.

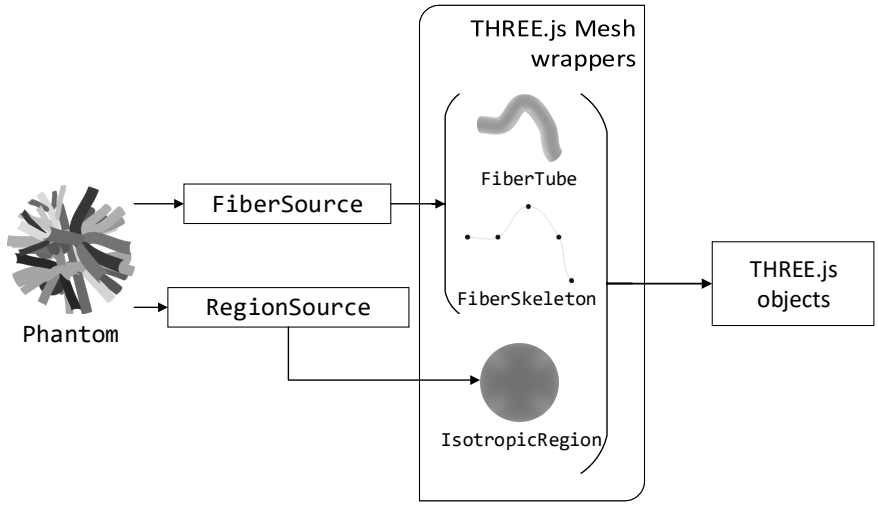



Figure 2.4: Four groups of objects interacting between them are responsible for representing phantom element: The Phantom object, Source classes, Mesh-wrapper classes and *THREE.js* objects.

In order to simplify the design, classes and objects were divided into 4 different levels, as shown in figure 2.4.

[2]In object-oriented programming, a class is a template from which new objects may be created.

1. **Phantom object**
   Its aim is to contain all references to the classes related to each element of the phantom and handle their modification. Although a class was created for future purposes, the application only contains one phantom object during its execution.

2. **Source classes**
   They contain the description of each element in the phantom. Regarding the fiber bundles, this class is also responsible for computing the trajectory of the fiber and supplying it to the mesh-wrapper class objects.

3. **Mesh-wrapper classes**
   The three such classes process the information contained in the source classes and compute *THREE.js* mesh objects ready to be added to the scene.

4. ***THREE.js* objects**
   Basic and necessary elements for creating a scene and rendering it in *WebGL*. These are objects whose prototype[3] is included in the *THREE.js* library, so they are the only classes in the phantom display design that have not been designed for this specific project.

In order to properly understand the transition from a phantom description to a completely functional and interactive scene, Source classes are presented first:

### Source classes

The source classes are those which contain the information describing the element concerned. There is one source class for fiber bundles, `FiberSource`, and another for isotropic regions, `IsotropicRegionSource`. In addition, `FiberSource` contains the methods[4] for computing the path of the fiber bundle.

In accordance with the phantom description files, source classes contain the same properties[5] as explained in 1.2.3. In addition, those may contain a color property with class `THREE.Color` as well. Although not compulsory when loading a phantom description file, when exported from the app, this property is included. This does not affect *Phantomαs*.

`FiberSource` class contains the method `polyCalc`. This method runs the code necessary to compute the coefficients of the following 3$^{\text{rd}}$-order polynomial describing the path:

$$f(t) = a + b\,\frac{t-t_i}{t_{i+1}-t_i} + c\left(\frac{t-t_i}{t_{i+1}-t_i}\right)^2 + d\left(\frac{t-t_i}{t_{i+1}-t_i}\right)^3$$

where $a$, $b$, $c$ and $d$ are the coefficients seeked by `polyCalc`, and $t$ is the "timestamp", a value over the unit relative to the length of the path.

[3] Code containing the class definition of an object

[4] Function that once defined is available to any of the objects of the same class, usually acting over its properties.

[5] Variables contained by default in all objects of the same class

Coefficient values are stored as properties and later used by the method `interpolate`, which allows *THREE.js*' objects to retrieve the trajectory from a time-stamp. A `setControlPoint` function is also available, which is responsible for taking all the steps necessary to recompute the path after changing the position of a control point.

### *THREE.js* objects

As explained before, for creating and managing the three dimensional environment representing the phantom, the *JavaScript* library *THREE.js* was used. In this section, the main classes used from this library are introduced.

The core of the representation is the scene, an instance of the `THREE.Scene` class. To set the view, a camera needs to be added. The camera chosen was `THREE.PerspectiveCamera`, as it offers the lesser deformation.

Once the scene and the point of view are placed, those need to be rendered by a renderer object. *THREE.js* contains several renderer classes. For *WebGL* rendering (see section 2.1), the corresponding class is `THREE.WebGLRenderer`. Renderer objects have a *DOM element*[6] for placing them in the page.

Every time the scene needs to be updated `renderer.render` must be executed. This function is placed in a function named `render`, called every time the scene suffers a change. When a continuous rendering is needed, a native *WebGL* function called `requestAnimationFrame` is given the rendering reference. This function can set the proper calling interval.

For showing the scene up, lights are also needed. In our specific scene, an ambient light `THREE.AmbientLight` was added and 8 directional lights placed in each corner of the space.

The code for a simplified example of the scene used in this specific application is shown in figure 2.5. The actual one is contained in the `show` function and can be found in the annex. Note that no object is present as only cameras and lights are added to the scene.

The main type of objects to be added to a scene are called "meshes", instances of `THREE.Mesh`. Those are the result of processing a geometry object and a material object. For all mesh elements shown in the scene, `THREE.MeshBasicMaterial` was used. Geometries may be created by using several classes present in the library.

In addition to the classes included in *THREE.js*, two other classes were added as another library, appended directly from *THREE.js*' source code [30]. These are `THREE.TrackballControls`, which allows the user to freely move around the scene by using mouse and touch gestures, and `THREE.TransformControls`, which builds an interactive interface over any mesh (in this case, fibers' control points) that allows free *drag-and-drop* to change position.

[6] Element appendable in an HTML page via *JavaScript* code.

```
1  var renderer = new THREE.WebGLRenderer();
2  document.appendChild(renderer.domElement);
3
4  var scene = new THREE.Scene();
5  var aspect = window.innerWidth / window.innerHeight;
6  var camera = new THREE.PerspectiveCamera(50, aspect, 1, 100));
7  scene.add(camera);
8
9  var ambientLight = new THREE.AmbientLight( 0xffffff, .5 );
10 var directionalLight = new THREE.DirectionalLight(0x555555, .15);
11
12 scene.add(ambientLight, directionalLight);
13
14 renderer.render(scene, camera);
```

Figure 2.5: Sample of the code present in `show` function, used for setting up the scene.

### Mesh-wrapper Classes

Once the source classes are defined, their information is taken to the *THREE.js* scene. This task is performed by the mesh-wrapper classes, which include a reference to their source object and return an object capable of being added directly to the scene.

In case their source object does not include a `color` property, mesh-wrapper classes do create it by assigning a random value obtained from the same color library used by `phantomas_view`, for display homogeneity. This library is stored in a constant named `colors`.

Whereas two different classes were defined for the source objects, there are three for the mesh-wrapper. As shown in figure 2.6, these are:

**a)** **`FiberTube`**: Generates the tube mesh, a `THREE.Mesh` object.

**b)** **`FiberSkeleton`**: Generates a path thread and spheres marking the control points. These are two objects of classes `THREE.Line` and `THREE.Mesh`.

**c)** **`IsotropicRegion`**: Generates the single sphere representing an isotropic region, as a `THREE.Mesh` instance.

The source objects built can be easily represented in a `THREE.Scene` by using the mesh-wrapper classes. Example 2.7 illustrates this procedure by assuming that two source instances, `fiber` and `region` were already built.

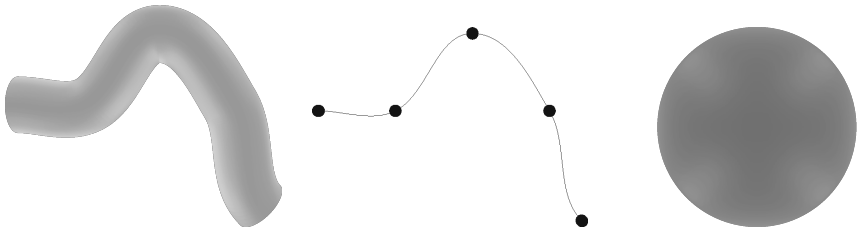



Figure 2.6: Meshes wrapped by mesh-wrapper classes. a. Fiber tube representation in `FiberTube`. b. Fiber path and control points in `FiberSkeleton`. c. `IsotropicRegion` representation.

```
1  // Our three variables are already defined
2  scene instanceof THREE.Scene;                // true
3  fiber instanceof FiberSource;                // true
4  region instanceof IsotropicRegionSource;     // true
5
6  var tube = new FiberTube(fiber);
7  scene.add(tube.mesh);
8
9  var skeleton = new FiberSkeleton(fiber);
10 scene.add(skeleton.line);
11 scene.add(skeleton.spheres);
12
13 var sphere = new IsotropicRegion(region);
14 scene.add(sphere.mesh);
```

Figure 2.7: Sample code to add phantom elements to the scene using mesh-wrapper classes.

### Phantom class

This is the last class present in the phantom display structure. It is the largest one as in addition to containing references to all source and mesh-wrapped objects, it also contains all the required methods to change the way the phantom shows up in the scene.

The first thing this class features is a reference to each of the source and mesh-wrapper objects, which are constructed by itself throught the method `Phantom.addFiber` or `Phantom.addIsotropicRegion`. Their structure is defined in figure 2.8.

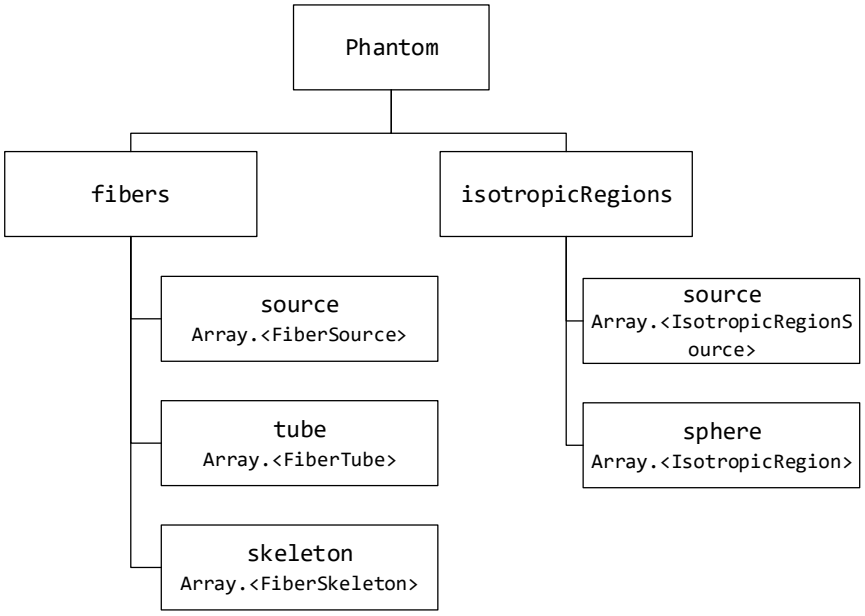



Figure 2.8: Structure of the main properties contained in `Phantom` object and its class.

Although not necessary for tweaking the display, the GUI requires the arrays to be the same length and the indexes to match between related elements.

The application was designed so the only needed object to change the visualization was the `Phantom` object. Thus, it contains several methods that allow this behavior, and which subsequently call any required procedure for taking it into action.

A simplified list of the most relevant `Phantom` methods may be found in table 2.1. For further description, please check the developer documentation (see page 45) or the code found in the annex.

| Method call | Description |
|---|---|
| `addFiber(fiber)` | Adds a Fiber to the Phantom, by creating their `FiberTube` and `FiberSkeleton`. |
| `addIsotropicRegion(region)` | Adds a Fiber to the Phantom, by creating its `IsotropicRegion`. |
| `addToScene(scene)` | Adds all Phantom bundles to the given scene. |
| `newFiber()` | Creates a new "blank" fiber in the scene. |
| `newIsotropicRegion()` | Creates a new "blank" isotropic region in the scene. |
| `addCP(fiberindex, cpbefore)` | Adds a new Control Point to a specified Fiber in the Phantom. |
| `removeCP(fiberindex, cp)` | Removes an existing Control Point of a specified Fiber in the Phantom. |
| `fadeAll(opacity)` | Fades all bundles to the given opacity. |
| `unfadeAll()` | Unfades all bundles. |
| `fiberHighlight(n)` | Fades all but the given fiber. |
| `cpHighlight(fiber, cp)` | Overlays a colored slightly bigger sphere over a control point. Used for forcing user focus onto it. |
| `revealSkeleton(scene, n)` | Adds Phantom to the scene and fades all by adding a Skeleton fiber to a given fiber. |
| `regionHighlight(n)` | Fades all but the given region. |

Table 2.1: Simplified list of most relevant `Phantom` methods.

### 2.4.3 Observer pattern design

In order to provide the consistency between all elements present in a phantom, the "observer pattern" was implemented in *Phantomαs* Web Designer. This pattern is used in software when an object is acting as an "observer", being dependent of another object, the "subject". When implemented, once the subject changes its state, all of its observers are notified about it [31]. Its behavior is schematized in figure 2.9.

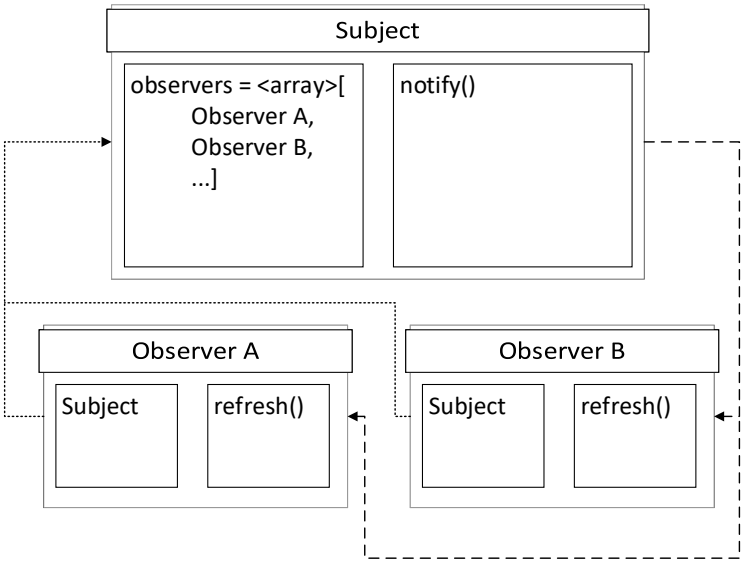



Figure 2.9: Schema representing the behavior of the observer pattern. Observers, referenced to a subject, are notified once `notify()` method is triggered.

During the development of this application, the observer pattern was used between source and mesh-wrapper objects. Source objects contain a method named `addObserver` for this purpose, whereas all mesh-wrapper objects have a `refresh` method.

By using any of the methods included in source classes, all observers get their `refresh` method called. This way ensures the concordance between these two core parts of the application.

### 2.4.4 Browser stability

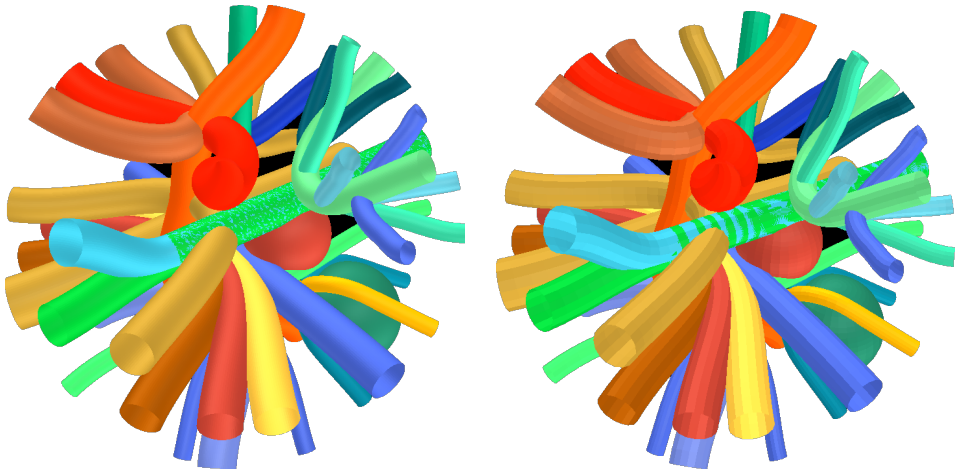
When designing an application to be run in several clients, stability is a point to be taken into account in order to ensure a suitable performance and user experience.

In *Phantomαs* Web Designer the *WebGL* canvas is the main resource spender. As so, we may lower them: as long as the phantom is displayed, we may disregard good graphic effects.

To keep a limited number of graphical elements to display we may specify the amount of segments meshes will feature. Segments are the number of vertices conforming a mesh. The lesser the segments are, the lesser the computing needed, making the application more fluent by lowering some of the displaying quality. Changing the amount of segments is purely visual and does not affect the phantom edition at all. This can be appreciated in figure 2.10.

The amount of segments in a mesh has to be specified in its building geometry. To handle this, mesh-wrapper constructors[7] expect an optional parameter as an object, which may include the amount of segments to build.

This parameter is determined by the `Phantom` function, which returns the optimal number out of a constraint list set in the constant `meshConstraints`, declared at the main script of the code.



a b

Figure 2.10: Applying segment constraints: a. Phantom with non-distinguishable segments. Experience was laggy. b. Phantom with constrained segments. Segments are visible but experience was smooth.

From several experience tests taken in different clients and platforms, the constant `meshConstraints` is currently specified in *Phantomαs* Web Designer as shown in figure 2.11.

[7]Functions that create an object of a specific class.

```
1   var meshConstraints = {
2   maxTotalAxialSegments: 1440,
3   maxMeshAxialSegments: 64,
4
5   maxTotalRadialSegments: 480,
6   maxMeshRadialSegments: 32,
7
8   maxTotalLineSegments: 960,
9   maxMeshLineSegments: 128,
10
11  maxTotalSkeletonSphereSegments: 240,
12  maxMeshSkeletonSphereSegments: 16,
13
14  maxTotalIsotropicRegionSegments: 1024,
15  maxMeshIsotropicRegionSegments: 32
16  }
```

Figure 2.11: Constant `meshConstraints` declaration. The number of segments specified in each of the rules is always into account by mesh-wrapper classes.

### 2.4.5 *HTML*

Web browsers work over *HTML* files. These contain the *HTML* markup itself, with both *JavaScript* and *CSS* code if applicable. This code may be either embedded in the *HTML* or referenced to an external file.

This application is wrapped in a single *HTML* file, named `phantomas.html` and located in the root directory. All *CSS* and *JavaScript* code is referenced.

The *HTML* page only contains the elements which remain static in the page, whereas the rest are managed by the GUI's *JavaScript* code.

To understand the structure of the *HTML* page, we must remind the structure of the interface itself, introduced in figure 2.3 (page 17). The same structure along with its *HTML* elements is reproduced in figure 2.12.

Each area is defined by a `<div>` element[8]. There are three main divisions, placed in three recognizable columns. Inside each, more `<div>` elements are placed to ease the interaction with *CSS* and *JavaScript* code. `<ul>` elements[9] are also used for element lists or for properly placing consecutive elements.

The *HTML* code structure is shown in figure 2.13. Those empty elements that are to be filled by *JavaScript* code are shown in grey. A simplified *HTML* code for building this interface can be found in figure 2.14.

[8]Defines a division in an *HTML* document. Groups elements to format them with *CSS*.
[9]Delimits an unordered list in an *HTML* document.

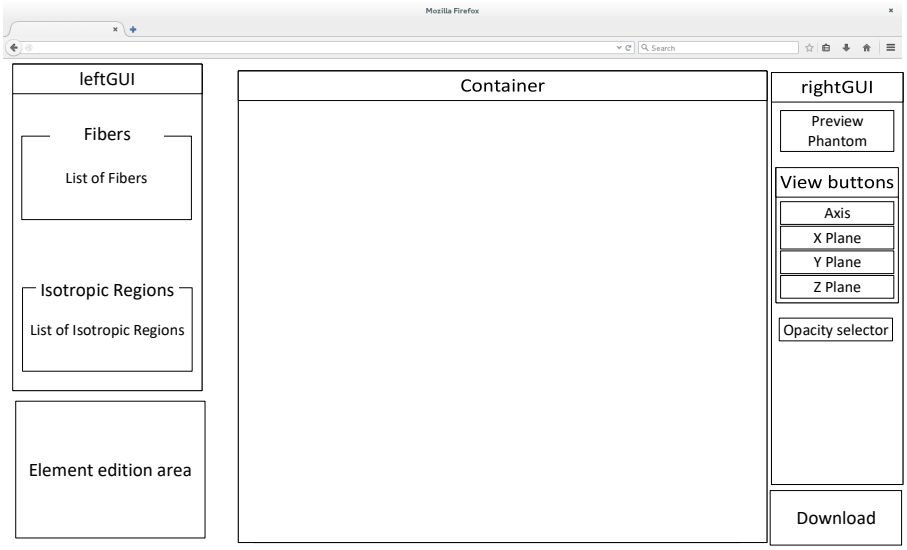



Figure 2.12: User interface schema as shown in the *HTML* page, drawn over the interface structure shown in figure 2.3.

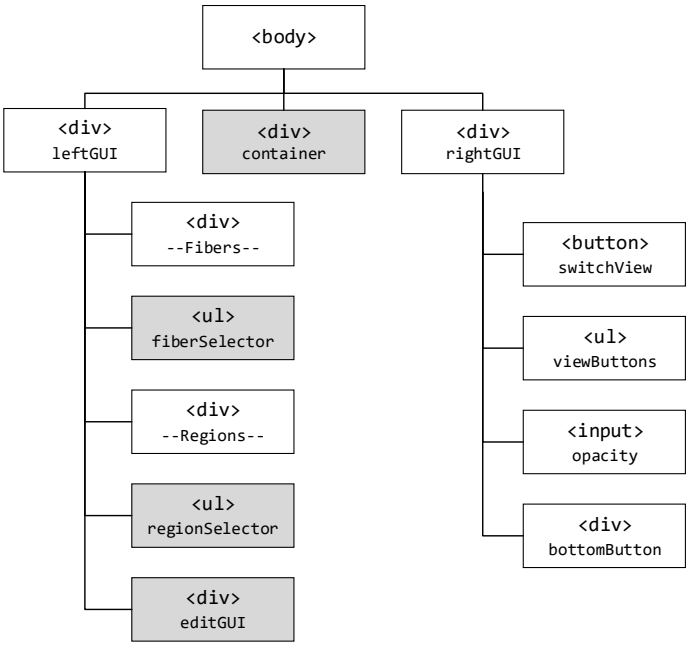



Figure 2.13: *HTML* code structure. Those elements shown in grey are created empty as will be managed by *JavaScript* GUI code.

```
<body>
  <div id="leftGUI">
    <div>
      Fibers
    </div>
    <ul id="fiberSelector">
    </ul>
    <br>
    <div>
      Isotropic Regions
    </div>
    <ul id="regionSelector">
    </ul>

    <div id="editGUI">
    </div>
  </div>

  <div id="container">
  </div>

  <div id="rightGUI">
    <button id='switchViewButton' />
    <br>
    <br>
    <ul>
      <li>
        <button id='toggleAxesButton'>
      </li>
      <li>
        <input type='button' title="X Plane (X)">
      </li>
      <li>
        <input type='button' title="Y Plane (Y)">
      </li>
      <li>
        <input type='button' title="Z Plane (Z)">
      </li>
    </ul>
    <br>
    <input id='opacitySelector'/>
    <div id="bottomButtons">
      <button title="Save (S)">
    </div>
  </div>
</body>
```

Figure 2.14: *HTML* simplified code for building the interface shown in figure 2.13.

### 2.4.6 GUI Construction

As an *HTML* file only contains the pattern to the General User Interface (GUI), it has to be built by *JavaScript* code. GUI Construction concerns those functions which read the information from the phantom and display the corresponding GUI.

There are three empty DOM Elements placed in the *HTML* page, awaiting content fill (see figure 2.13). Each of them is identified by an ID which *JavaScript* code may reference to, as seen in figure 2.14. Those are:

- Fiber selection list
- Isotropic Region selection list
- Edition user interface

There are seven different GUI Construction functions that act depending on user's actions. A list with their description can be found in table 2.2, and the timeline in figure 2.15 shows when are those fired.

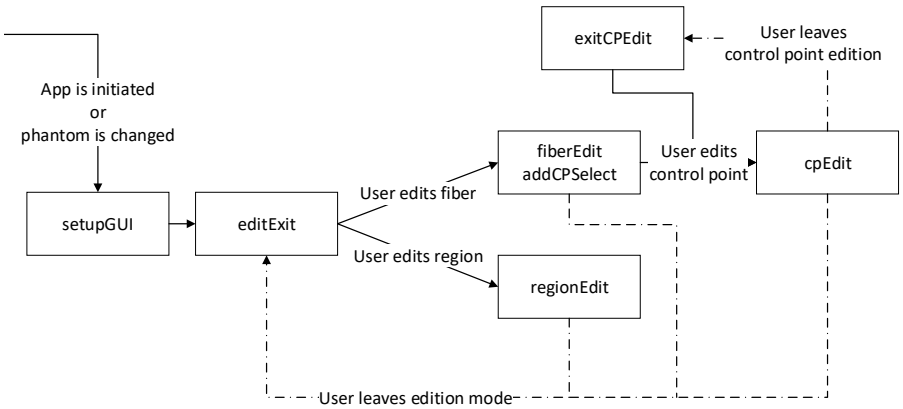



Figure 2.15: GUI Constructors' functions firing depending on actions taken by the user.

As seen in figure 2.15, the interface is initiated with both `setupGUI` and `editExit`. From then on, other GUI construction functions are recursively fired.

Each of the GUI construction functions are introduced as follows.

| Function | Description |
|---|---|
| `setupGUI()` | Constructs a basic-static GUI when no action has taken place yet. |
| `editExit()` | Removes any edition UI. Adds new element buttons. |
| `regionEdit(index)` | Adds the isotropic region edition GUI. |
| `fiberEdit(index)` | Adds the fiber edition GUI. |
| `addCPselect()` | Adds the control point selector UI for the current fiber. |
| `cpEdit(index)` | Constructs the Control Point edition UI for a given index of a control point. |
| `exitCPedit()` | Removes the former Control Point edition UI. |

Table 2.2: GUI construction function collection

#### `setupGUI` and `editExit`

Function `setupGUI` is fired along with `editExit` once the app is initiated or the phantom array of elements is changed.

The former is responsible for filling the element selector lists. The latter empties the edition `<div>` and builds new element buttons on top. This is the reason why it needs to be called after the app is initiated.

Each of these procedures are simple; `setupGUI` reads from `Phantom` object (see section 2.4.2 on page 23) the amount of objects present on it and builds one selection element for each of them.

A simplified code may be consulted in figures 2.16 and 2.17.

#### `fiberEdit` and `regionEdit`

These two functions are responsible for building the main properties' edition interface for the respective elements. They only require a single parameter, the index of the element in the `phantom` object array.

Main properties' edition interface consists in a form structure with information and editable elements.

For a fiber, these elements are

- Number of points
- Fiber color
- Path length
- Radius (editable)
- Tangents (editable)

```
// Retrieve DOM elements
var fiberSelector = document.getElementById("fiberSelector");
var regionSelector = document.getElementById("regionSelector");

// Add *none* option
var option = document.createElement("LI");
option.innerHTML = '*none*'
fiberSelector.appendChild(option);
regionSelector.appendChild(option);

// Add the rest of the options
phantom.fibers.source.forEach(function(fiber, index) {
  var option = document.createElement("LI");

  //Color mark for the selector
  var selectColorSpan = document.createElement("span");
  selectColorSpan.style.color = fiber.color.getStyle();
  selectColorSpan.innerHTML = '&#9632; '; //#9632 is the unicode
      reference for the black square

  //Text in selector
  var selectTextSpan = document.createElement("span");
  selectTextSpan.innerHTML = fiber.controlPoints.length.toString() + "
      points";

  option.appendChild(selectColorSpan);
  option.appendChild(selectTextSpan);

  fiberSelector.appendChild(option);
});

// Same for isotropic regions
phantom.regions.source.forEach(...
```

Figure 2.16: Simplified version of `setupGUI` function.

```
// Empty edit GUI
var editGUI = document.getElementById('editGUI');
editGUI.innerHTML = ""

var newfiberbutton = document.createElement("BUTTON");
newfiberbutton.innerHTML = "New Fiber";

var newregionbutton = document.createElement("BUTTON");
newregionbutton.innerHTML = "New Region";

editGUI.appendChild(newregionbutton);
editGUI.appendChild(newfiberbutton);
```

Figure 2.17: Simplified version of `editExit` function.

and for a region they are

- Region color
- Radius
- Position, containing:
  - Position in x (editable)
  - Position in y (editable)
  - Position in z (editable)

These elements are added in `editGUI <div>` element. In the case of `fiberEdit`, it is always called along with `addCPSelect`, which appends the list of control points for the specific fiber.

#### `addCPSelect`, `cpEdit` and `exitCPEdit`

Three functions are responsible for the control points edition interface. `cpEdit` builds the edition interface of a control point, while `exitCPEdit` empties its space.

Interface is built in a new `<table>` element created by `fiberEdit`. It contains the control point selection, created by the function `addCPSelect`. This function was designed separately to allow an independent refresh.

The table hosting the control point edition interface consists in two columns. The left one, thinner, contains the vertical list of the control points, while the second is the one where DOM elements responsible for the control point edition are built.

Elements present in a control point edition interface concern:

- Order number of the current control point
- Position in x (editable)
- Position in y (editable)
- Position in z (editable)
- Drag and drop toggle button
- Undo edition button
- New control point button
- Remove control point button

The *HTML* structure schema is summarised in figure 2.18, as contained in "editGUI" `<div>` element (see figure 2.13 in page 28).

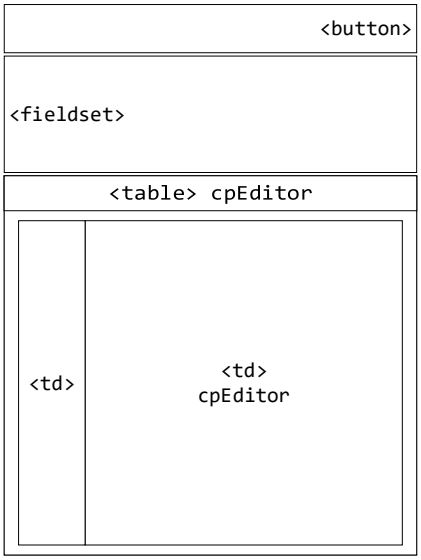



Figure 2.18: "editGUI" `<div>` element *HTML* structure during a control point edition.

### 2.4.7 GUI Handlers

GUI Handlers are all those functions called by a specific GUI element that has been set to do so when created.

Elements having a GUI Handler target have it referenced as an event. This reference is created when this element is declared, either by a GUI Constructor or in the *HTML* file.

There are several events to set in a DOM Element. In *Phantomαs* Web Designer the most used are specified in table 2.3.

| **Event** | **Description** |
|---|---|
| `onload` | Browser has finished loading the page |
| `onmouseenter` | Cursor is moved onto the element |
| `onclick` | User clicks the HTML element |
| `onchange` | HTML element has been changed |
| `onmouseleave` | Pointer is moved out of the element |
| `keyup` | User releases a keyboard key |

Table 2.3: Events used over application's DOM elements and their description [6].

All functions of the GUI handlers group are described in table 2.4. In addition, there is a small group of handlers related to *CSS* classes in the document. Those will be introduced in section 2.4.11.

In several parts of the code it may also be seen that functions were designed for being as well called to restore phantom display mode.

This is used by class `GuiStatus`, introduced in next section, does specific[10] calls to `fiberSelectClick`, `regionSelectClick` and `cpSelectClick` functions in order to restore phantom display mode.

### 2.4.8 GUI Managers

Last group of functions responsible for the GUI are the GUI Managers. Those harmonize the whole GUI with its environment.

The class `GuiStatus` is part of the GUI Managers. Its function is storing the task the user is performing and calling the `Phantom` class for visualization harmony. An schema of its functioning is shown in figure 2.19. Only one `GuiStatus` object is present at once.

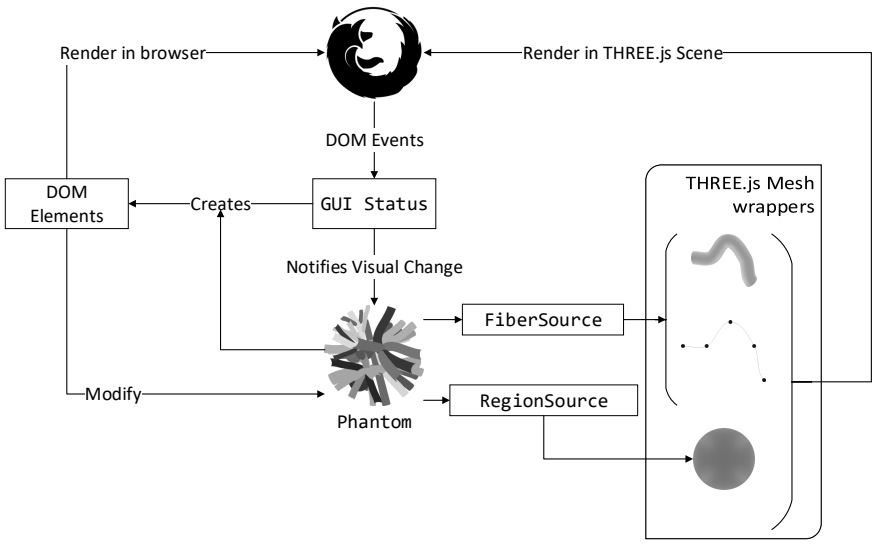



Figure 2.19: Class interaction in *Phantom∝s* Web Designer, with `GuiStatus` class having an essential function.

In a `GuiStatus` object the GUI current status is stored in five properties, as shown in table 2.5.

[10]Those calls are flagged with a boolean. When true, these functions act slightly different.

| Handler function | Description |
|---|---|
| `switchViewButton` | Handler for preview button. Switches on and off the fade of the scene. |
| `fiberSelectClick` | Events to be fired when a fiber was selected from the list. |
| `regionSelectClick` | Events to be fired when an isotropic region was selected from the list. |
| `cpSelectClick` | Events to be fired when a control point was selected from the list. |
| `newFiberClick` | Fires the creation of a new fiber and goes into edition. |
| `newIsotropicRegionClick` | Fires the creation of a new isotropic region and goes into edition. |
| `removeFiberClick` | Fires the removal of a fiber and quits edition. Prompts the user for confirmation. |
| `removeIsotropicRegionClick` | Fires the removal of an isotropic region and quits edition. Prompts the user for confirmation. |
| `newCPclick` | Fires the addition of a new Control Point after the current one. Gets into edit mode. |
| `newCPonmouseover` | Hover for new control point button. Simulates to the scene the addition of a new control point in green color. |
| `newCPonmouseout` | Restores the scene after un-hover in the new control point button. |
| `removeCPclick` | Fires the removal of a control point and quits edition. Prompts the user for confirmation. |
| `toggleAxes` | Toogles axes view button. Switches between showing or removing them in the scene. |
| `moveCameraXY` | Moves view to the XY plane. |
| `moveCameraXZ` | Moves view to the XZ plane. |
| `moveCameraZY` | Moves view to the ZY plane. |
| `opacitySelectChange` | Fired when the value in the opacity selector is changed. Corrects the value and fires the scene change. |
| `saveClick` | Prompts the user to download the description of the current phantom. |

Table 2.4: List of GUI Handler functions present in the code.

| Name | Type | Default value | Description |
|---|---|---|---|
| `editingFiber` | Number | `undefined` | Index of currently being edited fiber. If any, `undefined`. |
| `editingCP` | Number | `undefined` | Index of currently being edited control point. If any, `undefined`. |
| `editingRegion` | Number | `undefined` | Index of currently being edited isotropic region. If any, `undefined`. |
| `previewing` | Boolean | `false` | Whether preview mode is active or not. |
| `dragAndDropping` | Boolean | `false` | Whether drag and drop control point edit mode is active or not. |

Table 2.5: Properties in `GuiStatus` object with their type and default value.

`GuiStatus` features two main methods. First, `editing`, which makes it to change its properties to the given status. Second, `retrieve`, which makes the `Phantom` object and the GUI *HTML* elements to restore their default displaying mode by reading its own properties. `retrieve` uses several GUI Handlers' functions.

Function `resizeGUI` is the second and last element of the GUI Managers. Its function is to make all the elements fit in the screen so that no window scroll bar pops up.

The way to do so is by re-sizing the element selectors, leaving the exact amount of shown elements so that all the elements fit in the `leftGUI <div>`. Other elements in the selection lists can be seen by scrolling.

`resizeGUI` is called every time the browser window is re-sized or other elements are created in any of the `leftGUI <div>` areas.

### 2.4.9 JSON load and save

As this app was developed for avoiding plain text edition of *Phantomαs*' JSON files, functions for loading and exporting them are essential. These are `loadPhantom` and `pushDownload`.

A simple string containing the JSON information is all that `loadPhantom` needs to load the file. It creates all the source objects and returns all of them added in a `Phantom` object. A sample code of this function can be seen in figure 2.20.

```
1  function loadPhantom( string ) {
2    // Create an empty phantom
3    var phantom = new Phantom();
4    // Parse objects contained in the string
5    var loadedFibers = JSON.parse(string).fiber_geometries;
6    var loadedRegions = JSON.parse(string).isotropic_regions;
7
8    // Add them to the phantom
9    for (var property in loadedFibers) {
10     if (loadedFibers.hasOwnProperty(property)) {
11       var fiber = loadedFibers[property.toString()];
12       // addFiber Phantom method is used
13       phantom.addFiber(new FiberSource(fiber.control_points, fiber.
             tangents, fiber.radius);
14     }
15   }
16   for (var property in loadedRegions) {
17     ...
18   }
19
20   // log an error in case fibers or regions were not found.
21   if (phantom.isotropicRegions.source.length == 0) console.warn('Any
        region found in file');
22   if (phantom.fibers.source.length == 0) console.warn('Any fiber found in
        file');
23
24   return phantom;
25 }
```

Figure 2.20: Simplified code contained in the *loadPhantom* function, which creates a new `Phantom` object with all of its elements out of a parsed JSON string.

Creating a string with the contents of a JSON file just involves turning into a string each of the properties of the elements in the phantom. The method `export` contained in `Phantom` prototype returns this string.

Note that it also adds the `color` property, which `loadPhantom` is also expecting. In case there is none, it is ignored and randomized by mesh-wrapper constructors.

As for the name contained in *Phantomαs*' JSON files, those are set just with a single number, which is the index in the `Phantom` array.

Allowing the user to download the JSON file is somewhat more complex. For allowing the download of a file which does not exist in the server, a redirection to an URL which includes its data was created [32].

First, an empty `<a>` element is placed somewhere in the page, styled to not be displayed:

```
<a id="downloadAnchorElem" style="display:none"/>
```

Function `pushDownload` encodes the JSON string as a link redirection, adds the file name and simulates its click. A simplified code of the `pushDownload` function is shown in figure 2.21.

The file name includes a time stamp. For instance, a file downloaded on 12/05/2017 at 18:46h would be named after `120520171846-phantom_save.json`.

```
1  // Encode the JSON string "content"
2  var uriContent = "data:text/json;charset=utf-8," + encodeURIComponent(
       content);
3
4  // Find the empty element located in the HTML page.
5  var dlAnchorElem = document.getElementById('downloadAnchorElem');
6  // Add the encoded string as the href attribute
7  dlAnchorElem.setAttribute("href", uriContent);
8  // Set the download filename. timestamp function creates a string in a
       ddmmyyyyhhmm structure
9  dlAnchorElem.setAttribute("download", timestamp()+"-phantom_save.json");
10 // Simulate a click in the anchor element
11 dlAnchorElem.click();
```

Figure 2.21: Code in *pushDownload* function responsible for pushing the download of a phantom description JSON file.

### 2.4.10 App initiation

Once the *HTML* page is loaded and the scripts referenced to it are loaded, `document.onload` function is executed. In this app, the `init` function was assigned as `document.onload`. It loads the JSON file (if any) and calls the main *JavaScript* functions.

#### *HTTP Request*

To load an specified JSON file, an *HTTP Request* is performed. This is included in the `init` function.

Due to further development expected in *Phantomαs*, for now the JSON file must be placed in the server and the user has no option to select it from its hard drive. The path to this file is to be specified in the URL itself, separated by a question mark. For example, for loading the file placed in path `examples/fibers.json` the URL would be:

```
phantomas.html?examples/fibers.json
```

Once the *JavaScript* function has located the file, it parses its contents and loads it into a `Phantom` object called `phantom`, by using the `loadPhantom` function. In case no file is specified or the file is not found, it enters in "scratch mode" by creating an empty `Phantom` object.

After having defined `phantom`, the app is initated by running both the `show` and `setupGUI` functions.

All the information regarding the HTTP request is logged in the *JavaScript* browser console for aiding at debugging. The code contained in `init` function is shown in figure 2.22.

### 2.4.11 *CSS*

Styling sheets in *Phantomαs* Web Designer do not only define the look of the *HTML* elements but also their style behavior when interacting between them.

Up to four *CSS* files are referenced in the *HTML* `<head>` section:

- **`main.css`**: Main styling sheet for the *HTML* elements. Defines main classes and their behavior.
- **`icons.css`**: Defines a class[11] pointing at a secondary file font containing the icons shown in buttons.
- **`w3.css`**: *W3C CSS* library for buttons styling.
- **`normalize.css`**: *normalize.css CSS* sheet, used for maintaining homogeneous appearance across different browsers (see section 2.2).

[11] A class in *CSS* is a group in which elements being part of it share the same styling. Elements usually change their class dynamically.

```
1  // Set init as window.onload function
2  window.onload = init;
3
4  function init() {
5    // Check whether a path was or not specified
6    if (location.href.indexOf('?') > 0) {
7      path = location.href.substring( location.href.indexOf('?') + 1);
8      makeRequest();
9    } else {
10     phantom = new Phantom();
11     console.log('No specified path found. Loading scratch mode.');
12     show();
13     setupGUI();
14   }
15
16   function makeRequest() {
17     var request = new XMLHttpRequest();
18     request.overrideMimeType("text/plain");
19     request.open("get", path, true);
20     request.onreadystatechange = function() {
21       if (this.readyState === 4) { // If request is completely finished
22         if (this.status === 200) { // If it was successful
23           phantom = loadPhantom(this.response);
24         } else { // If it was not successful
25           console.error('Error: ' + path + ' was not found. Loading
                 scratch mode.')
26           phantom = new Phantom();
27         }
28         show();
29         setupGUI();
30       }
31     };
32     request.send(null); // End the request
33   }
34 }
```

Figure 2.22: Code contained in the `init` function, responsible for loading the file request and executing the main initiation functions.

**main.css**

Main styling sheet `main.css` was built from scratch for this specific application. It serves three functionalities: placing correctly `<div>` elements in the page, styling *HTML* interface, and defining classes for the behavior of element selection lists during interaction.

The main goal of this application is that the interface remains static. This means that no scrolling bar pops up at any time in the window to avoid undesired behavior. This is ensured by precisely setting the margins in global `<html>` element and `<div>` divisors, all set with the `inline-block` display mode. The code used in `leftGUI <div>` is shown as a sample in figure 2.23.

```
1  hml, html, body {
2    margin: 0;
3    padding: 0;
4  }
5
6  #leftGUI {
7    float: left;
8    text-align: left;
9    display: inline-block;
10
11   width: 19%;
12   margin: 0;
13   padding: .5%;
14 }
```

Figure 2.23: Code used in `main.css` file for organizing the page display for the global `<html>` element and `leftGUI <div>`.

Setting up an harmonious look with the rest of the page for those generic DOM elements used in the GUI is also an aim of `main.css`. This involves `<input>` fields or `<ul>` and `<li>` elements used for ordering buttons and lists.

The action taken is setting background black and font white, as the *Three.js* scene, and disabling padding and bullets in lists as none is expected.

Last but really important function of `main.css` is styling element lists according to the user action. This makes a big leap in user experience.

This ability is managed by using *CSS* classes declared in `main.css`, which are set by *JavaScript* code in GUI Builders, and changed dynamically by a special category in GUI Handlers functions which focus just on elements' style class. These different classes are declared in `main.css` as shown in figure 2.24. GUI Handlers' functions managing those over elements may be consulted in table 2.25.

```
.enabledList {
  overflow: scroll;
  overflow-x: hidden;
  cursor: pointer;
  padding-left: 5px;
  border-style: solid;
  border-width: 1px;
  border-color: grey;
  border-right: 0;
}
.disabledList {
  background-color: grey;
  padding-left: 10px;
}
.optionUnselected {
}
.optionSelected {
  padding-left: 15px;
  color:yellow;
  font-weight:bold;
}
.optionOnMouseOver {
  font-weight:bold;
  background-color: DimGrey;
}
.optionSelectedAndOnMouseOver {
  background-color: DimGrey;
  padding-left: 20px;
  color:yellow;
  font-weight:bold;
}
```

Figure 2.24: Code used in `main.css` file for declaring classes used in dynamic GUI behavior.

```
function optionOnMouseOver(option) {
  if (option.className == 'optionSelected') {
    option.className = 'optionSelectedAndOnMouseOver';
  } else if (option.className == 'optionUnselected') {
    option.className = 'optionOnMouseOver';
  }
}
function optionOnMouseLeave(option) {
  if (option.className == 'optionSelectedAndOnMouseOver') {
    option.className = 'optionSelected';
  } else if (option.className == 'optionOnMouseOver') {
    option.className = 'optionUnselected';
  }
}
```

Figure 2.25: Two of the functions present in GUI Style Handlers, set as events and responsible of their look depending on user's actions.

#### `w3.css` and `icons.css`

Two stylesheets add more classes to the page. These are `w3.css` and `icons.css` and are used solely on buttons.

*World Wide Web Consortium (W3C)* deployed *W3.CSS*, a free-to-use styling sheet with the aim to provide a cross-platform good-looking and light-weight environment [33].
Its button classes were the only used to provide a better user experience.

Button values are shown as icons. This is thanks to a styling class defined in `icons.css` that calls a new font, stored in a `.woff` file in `icons/` root folder.
This font is custom built for reducing its weight, and contains 5 "Material Icons" [34], the ones shown in the GUI. Those are invoked by defining the class and writing the code assigned to the icon in the font. `icons.css` code is shown in figure 2.26.

```
@font-face {
  font-family: 'appicons';
  font-style: normal;
  font-weight: 400;
  src: url(../icons/icons.woff) format('woff');
}

.icons {
  font-family: 'appicons';
  font-weight: normal;
  font-style: normal;
  font-size: 24px;
  line-height: 1;
  letter-spacing: normal;
  text-transform: none;
  display: inline-block;
  white-space: nowrap;
  word-wrap: normal;
  direction: ltr;
}
```

Figure 2.26: Code used in `icons.css`, which defined a new font and a new styling class for invoking icons contained in custom font `icons.woff`.

#### `normalize.css`

As introduced in "Technologies involved" (page 11), *normalize.css* was used as a tool for harmonizing the style along most browsers in the market.

*CSS* stylesheet `normalize.css` is a slightly modified version of *normalize.css* [23] version 5.0.0. These modifications were taken during the development of the GUI for better responsiveness.

### 2.4.12 Homepage

To wrap a few sample examples in *Phantomαs* and referencing some contents related to the application, a plain *HTML* page was built.

Links referencing to elements such as the documentation, examples or information related to *Phantomαs* were included to guide the user through the possibilities the application offers.

This homepage currently looks as in figure 2.27. It is stored as `index.html` and contains no *JavaScript* nor *CSS* code apart from inline styling.

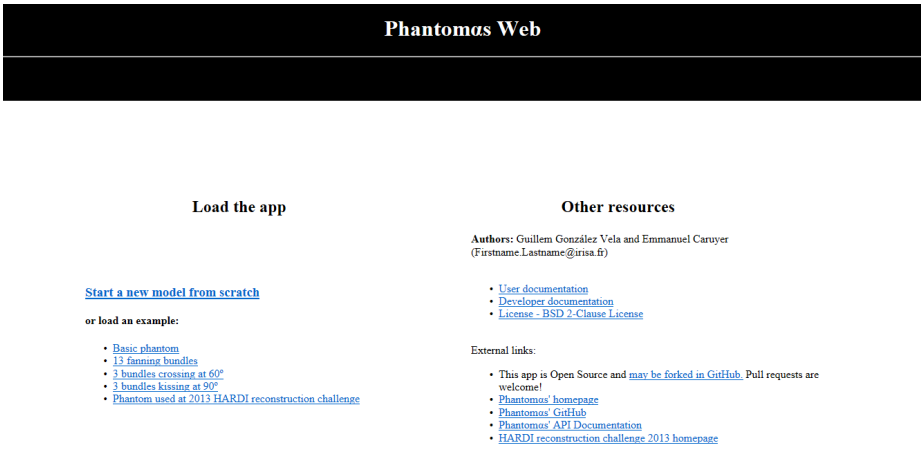



Figure 2.27: General look of simple *Phantomαs* Web Designer's homepage.

### 2.4.13 Documentation

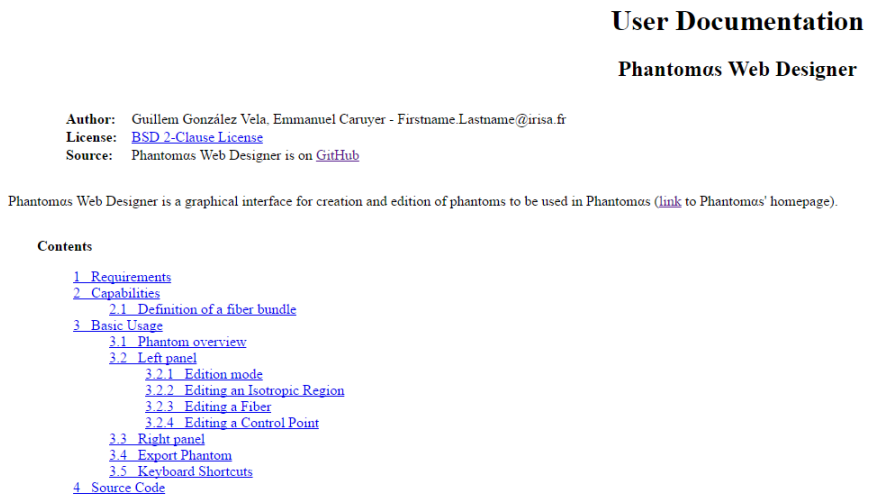
Two different documentations were created for *Phantomαs* Web Designer. The first is the "User Documentation", addressed to *Phantomαs* Web Designer's users. The other, the "Developer Documentation", is addressed to those developers who want to improve the code or build something related to it.

Both are rendered in *HTML*, so it may be read by using the same browser in which the application is being executed in.

In the "User Documentation" all possibilities regarding the usage of the app are explained. It is only directed towards the users.

It was created with *reStructuredText*, a lightweight markup language that renders in *HTML* [26]. A sample of the user documentation can be seen in figure 2.28.

As for as the "Developer Documentation" concerns, it was created with *JSDoc3*, an open-source API documentation generator for *JavaScript* [25].



**User Documentation**

**Phantomαs Web Designer**

**Author:** Guillem González Vela, Emmanuel Caruyer - Firstname.Lastname@irisa.fr
**License:** BSD 2-Clause License
**Source:** Phantomαs Web Designer is on GitHub

Phantomαs Web Designer is a graphical interface for creation and edition of phantoms to be used in Phantomαs (link to Phantomαs' homepage).

**Contents**

1 Requirements
2 Capabilities
2.1 Definition of a fiber bundle
3 Basic Usage
3.1 Phantom overview
3.2 Left panel
3.2.1 Edition mode
3.2.2 Editing an Isotropic Region
3.2.3 Editing a Fiber
3.2.4 Editing a Control Point
3.3 Right panel
3.4 Export Phantom
3.5 Keyboard Shortcuts
4 Source Code

**1 Requirements**

Phantomαs Web Designer was tested on Mozilla Firefox. Although, it is fully compatible with any modern internet navigator.

No extra software is needed.

**2 Capabilities**

Figure 2.28: General look of the heading of *Phantomαs* Web Designer's user documentation.

Each of the files, functions and classes to be documented have a heading of commented code between the flags `/**` and `*/`. Those flags are parsed and interpreted by *JSDoc3*, which creates a series of *HTML* files containing the whole documentation and the parsed code, all linked in between them.

An example of a heading with comments to be parsed by *JSDoc3* and its result can be found in figure 2.29.

In order to be consistent in each *JSDoc3* processing, a specific configuration file was created. This configuration is expected by the tool and was designed by following its documentation [25]. It is also given along with the application's code. This configuration file is formatted in JSON and its content may be consulted in figure 2.30.

```
1  function FiberSource(controlPoints, tangents, radius, color) {
2    /** @class FiberSource
3      * @classdesc A fiber bundle in Phantomas is defined as a cylindrical
           tube wrapped around
4      its centerline. The centerline itself is a continuous curve in 3D, and
            can be
5      simply created from a few control points. All the fibers created are
           supposed to connect two
6      cortical areas.<br>
7      * FiberSource is the basic Class for the representation of a Fiber.
           Objects containing
8      the geometries to be added to the scene are to be referred to
           FiberSource for
9      gathering any necessary information.
10
11     * @param {array} controlPoints Array-of-arrays (N, 3) containing the 3
           -D coordinates
12     of the fiber Control Points.
13     * @param {string} [tangents='symmetric'] Way the tangents are to be
           computed.
14     Available options: 'incoming', 'outgoing', 'symmetric'
15     * @param {number} [radius=1] Fiber radius; same units as
           controlPoints.
16     * @param {number} [color] Color in which the fiber should be
           displayed. If not
17     specified, to be picked randomly from {@link colors}.
18
19     * @property {array} observers Objects to be notified when some change
           is applied
20     */
```

Class: FiberSource

**FiberSource**

A fiber bundle in Phantomas is defined as a cylindrical tube wrapped around its centerline. The centerline itself is a continuous curve in 3D, and can be simply created from a few control points. All the fibers created are supposed to connect two cortical areas.
FiberSource is the basic Class for the representation of a Fiber. Objects containing the geometries to be added to the scene are to be referred to FiberSource for gathering any necessary information.

**Constructor**

`new FiberSource(controlPoints, tangents_opt, radius_opt, color_opt)`

Parameters:

| Name | Type | Attributes | Default | Description |
|---|---|---|---|---|
| controlPoints | array | | | Array-of-arrays (N, 3) containing the 3-D coordinates of the fiber Control Points. |
| tangents | string | <optional> | 'symmetric' | Way the tangents are to be computed. Available options: 'incoming', 'outgoing', 'symmetric' |
| radius | number | <optional> | 1 | Fiber radius; same dimensions as controlPoints. |
| color | number | <optional> | | Color in which the fiber should be displayed. If not specified, to be picked randomly from colors. |

Properties:

| Name | Type | Description |
|---|---|---|
| observers | array | Objects to be notified when some change is applied |

Source: js/FiberSource.js, line 17

Figure 2.29: Heading of the `FiberSource` class constructor and its result after *JSDoc3* processing.

```
{
  "source": {
    "include": ["js", "gui"],
    "includePattern": ".+\\.js(doc|x)?$",
    "excludePattern": "(^|\\/|\\\\)_"
  },
  "opts": {
    "destination": "./doc/developer/"
  },
  "tags": {
    "allowUnknownTags": true,
    "dictionaries": ["jsdoc","closure"]
  },
  "plugins": [],
  "templates": {
      "cleverLinks": true,
      "monospaceLinks": false
  }
}
```

Figure 2.30: Contents of *JSDoc3* configuration file, `jsdoc-conf.json`

### 2.4.14 File structure

All files containing the described *JavaScript* functions (extension `.js`) and *CSS* files (extension `.css`) are placed in a root folder which also contains the `.html` file of *Phantomαs*. Their relative path is placed in the *HTML* file heading, so that they can be loaded by the browser.

To avoid *JavaScript* code from running when dependent elements have not been loaded yet, the `window.onload` function is referenced, as explained in section 2.4.10.

Many other files are present in the source code folder, such as documentation or icons. The complete structure is shown in figure 2.31. The contents of each of the main files is explained in table 2.6, while the code present in the *HTML* `<head>`, for referencing those, may be consulted in figure 2.32.

```
phantomas-web/
├── index.html
├── phantomas.html
├── css/
│   ├── icons.css
│   ├── main.css
│   ├── normalize.css
│   └── w3.css
├── doc/
│   ├── jsdoc-conf.json
│   ├── developer/
│   │   └── index.html
│   └── user/
│       ├── img/
│       ├── source.rst
│       └── index.html
├── examples/
│   ├── 3Dfanning_13bundles.json
│   ├── 60crossing_3bundles.json
│   ├── 90kissing_3bundles.json
│   ├── fibers.json
│   └── isbi_challenge_2013.json
├── gui/
│   ├── cpedit.js
│   ├── fiberedit.js
│   ├── handlers.js
│   ├── regionedit.js
│   ├── resize.js
│   ├── setup.js
│   ├── status.js
│   └── stylehandlers.js
├── icons/
│   ├── icons.woff
│   └── favicon.ico
├── js/
│   ├── FiberSource.js
│   ├── MeshSource.js
│   ├── axes.js
│   ├── load.js
│   ├── main.js
│   ├── Phantom.js
│   ├── save.js
│   └── dragAndDrop.js
└── lib/
    ├── TrackballControls.js
    ├── three.min.js
    └── TransformControls.js
```

Figure 2.31: Structure of main files and folders present in root folder. Description of each available in table 2.6.

| File name | Description |
|---|---|
| `phantomas-web/` | Root folder |
| `index.html` | *Phantomαs* Web Designer's homepage |
| `phantomas.html` | *Phantomαs* Web Designer's HTML |
| `css/` | CSS files are placed in this directory |
| `icons.css` | Class definition for button icons |
| `main.css` | Main classes definition |
| `normalize.css` | *normalize.css* CSS file |
| `w3.css` | W3CSS button classes |
| `doc/` | Documentation |
| `jsdoc-conf.json` | JSDoc3 configuration file |
| `developer/` | Developer documentation |
| `index.html` | Main page for documentation, followed by its files |
| `user/` | User documentation |
| `img/` | User documentation image files |
| `source.rst` | User documentation reStructuredText source file |
| `index.html` | User documentation processed HTML file |
| `examples/` | *Phantomαs*' examples |
| `gui/` | GUI *JavaScript* code |
| `cpedit.js` | `cpEdit` and `exitCPedit` functions |
| `fiberedit.js` | `fiberEdit` function |
| `handlers.js` | GUI Handlers functions |
| `regionedit.js` | `regionEdit` function |
| `resize.js` | `resizeGUI` function |
| `setup.js` | `guiSetup` and `editExit` functions |
| `status.js` | GuiStatus class |
| `stylehandlers.js` | GUI Style Handlers functions |
| `icons.woff` | Font containing GUI's icons |
| `favicon.ico` | Page browser icon |
| `js/` | Core *JavaScript* code |
| `FiberSource.js` | `FiberSource` and `IsotropicRegionSource` classes |
| `MeshSource.js` | Mesh-wrapper classes |
| `axes.js` | `buildAxes` function |
| `load.js` | `loadPhantom` function |
| `main.js` | Variable declaration. `init` and `show` functions |
| `Phantom.js` | `Phantom` class |
| `save.js` | `Phantom.export` method, `pushDownload` function |
| `dragAndDrop.js` | `dragAndDrop` function |
| `lib/` | Libraries |
| `TrackballControls.js` | `THREE.TrackballControls` class |
| `three.min.js` | *THREE.js* r84 |
| `TransformControls.js` | `THREE.TransformControls` class |

Table 2.6: List of main files and folders present in root folder, as shown in figure 2.31, and its description.

```
<link rel="icon" href="icons/favicon.ico">
<link rel="stylesheet" href="css/normalize.css" />
<link rel="stylesheet" href="css/w3.css" />
<link rel="stylesheet" href="css/main.css" />
<link rel="stylesheet" href="css/icons.css">

<script type="text/javascript" src="lib/three.min.js"></script>
<script type="text/javascript" src="lib/TrackballControls.js"></script>
<script type="text/javascript" src="lib/TransformControls.js"></script>

<script type="text/javascript" src="js/FiberSource.js"></script>
<script type="text/javascript" src="js/MeshSource.js"></script>
<script type="text/javascript" src="js/Phantom.js"></script>
<script type="text/javascript" src="js/dragAndDrop.js"></script>
<script type="text/javascript" src="js/load.js"></script>
<script type="text/javascript" src="js/save.js"></script>
<script type="text/javascript" src="js/axes.js"></script>
<script type="text/javascript" src="js/main.js"></script>

<script type="text/javascript" src="gui/handlers.js"></script>
<script type="text/javascript" src="gui/resize.js"></script>
<script type="text/javascript" src="gui/status.js"></script>
<script type="text/javascript" src="gui/setup.js"></script>
<script type="text/javascript" src="gui/stylehandlers.js"></script>
<script type="text/javascript" src="gui/fiberedit.js"></script>
<script type="text/javascript" src="gui/regionedit.js"></script>
<script type="text/javascript" src="gui/cpedit.js"></script>
```

Figure 2.32: *HTML* `<head>` extract, which references necessary *JavaScript* and *CSS* code. These are referenced following the relative paths shown in figure 2.31.

## 2.5 Licensing and source

*Phantomαs* Web Designer is open-source software, hold by the *BSD 2-Clause License* ("Berkeley Software Distribution License"). This license allows modification and commercial use while always reproducing its content and acknowledging its author.

Source code is also open since its early development, as its *git* repository is public in *GitHub* [35], as seen in figure 2.33.

All of its future development is expected to continue through this same repository, in which pull requests[12] are welcome. The complete text of the license can also be found in the repository.

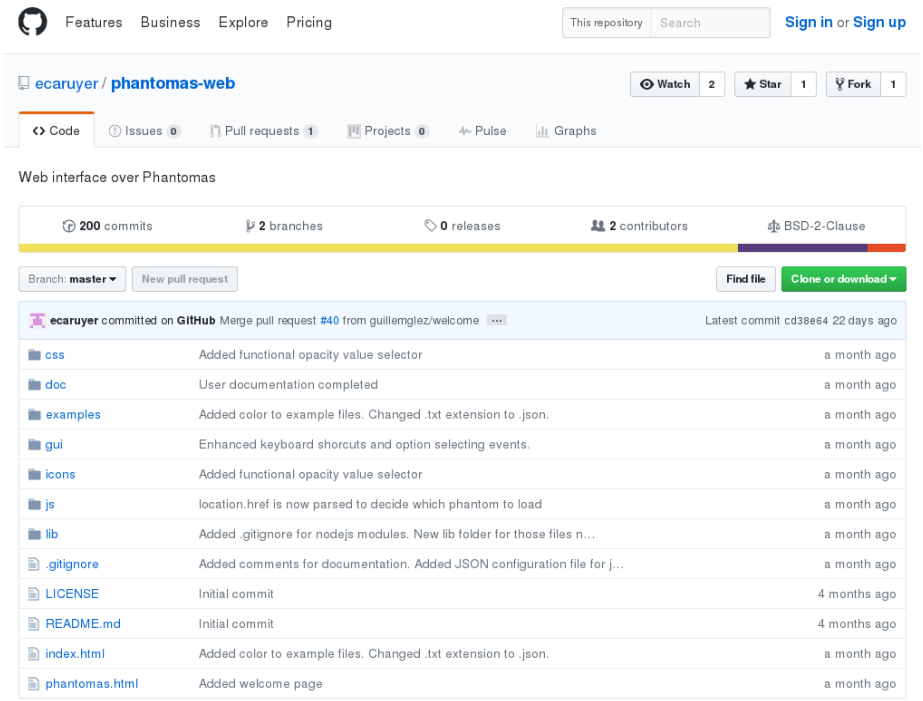



Figure 2.33: *Phantomαs* Web Designer's *GitHub* repository as of 2017-05-14.

[12] Source code modification suggested by another *GitHub* user.

# Chapter 3

# Results

In this chapter the result of the finished project from the user eye is presented.

Although for a full demonstration on how the software works and the way it meets its requirements a dynamic mean might be necessary, a sneak peek will be introduced to get the idea on how the application behaves.

The last version of *Phantomαs* Web Designer is hosted[1] in the URL `phantomas.ddns.net` and ready to use. Its source code is available in its *GitHub* repository `github.com/ecaruyer/phantomas-web`. Although the code itself does not need to be compiled, a local *HTTP* server is usually needed for loading local phantom description files.

### Software operation

The main requirements set were the ability of the software to load *Phantomαs* JSON files and being able to display the phantom contained in a 3D interactive environment. This functionality is presented in figure 3.1, by loading 2013 HARDI ISBI Challenge's phantom description [16].

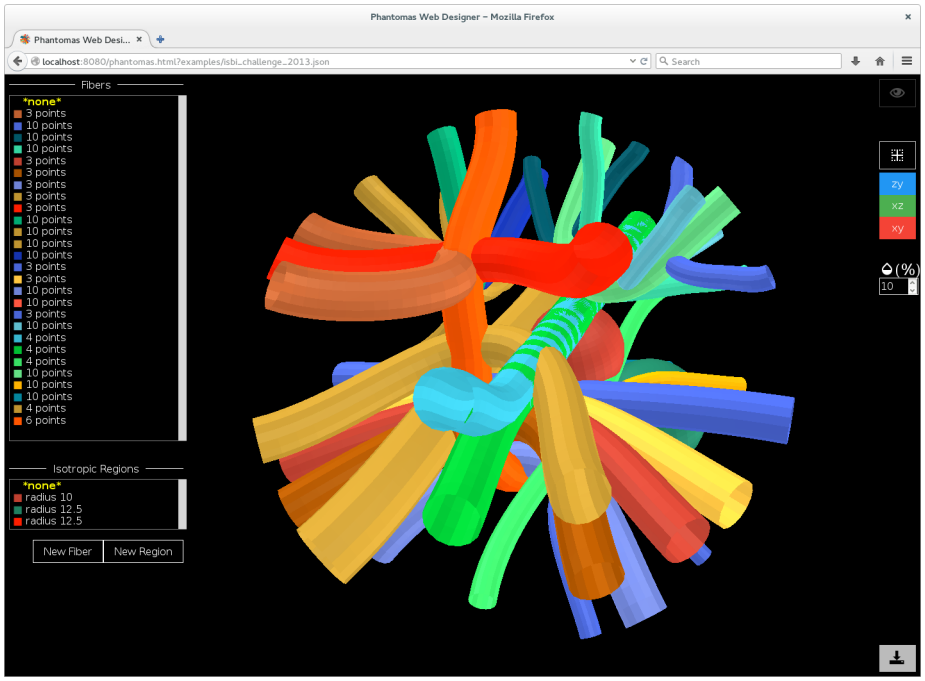



Figure 3.1: *Phantomαs* Web Designer in non-edit mode, displaying a 30-element phantom over *Mozilla Firefox 37* in a *Fedora 21* system.

As of the ability to recognize and analyze different elements, the application fades the one in which the mouse is placed over, allowing the user to easily recognize it. This feature can be seen in figure 3.2.

[1] Until 2017 fall

To aid the edition, it keeps focused while editing, although it might be unfaded at any moment – without affecting the edition process – by using the preview button placed at the top right corner of the screen.

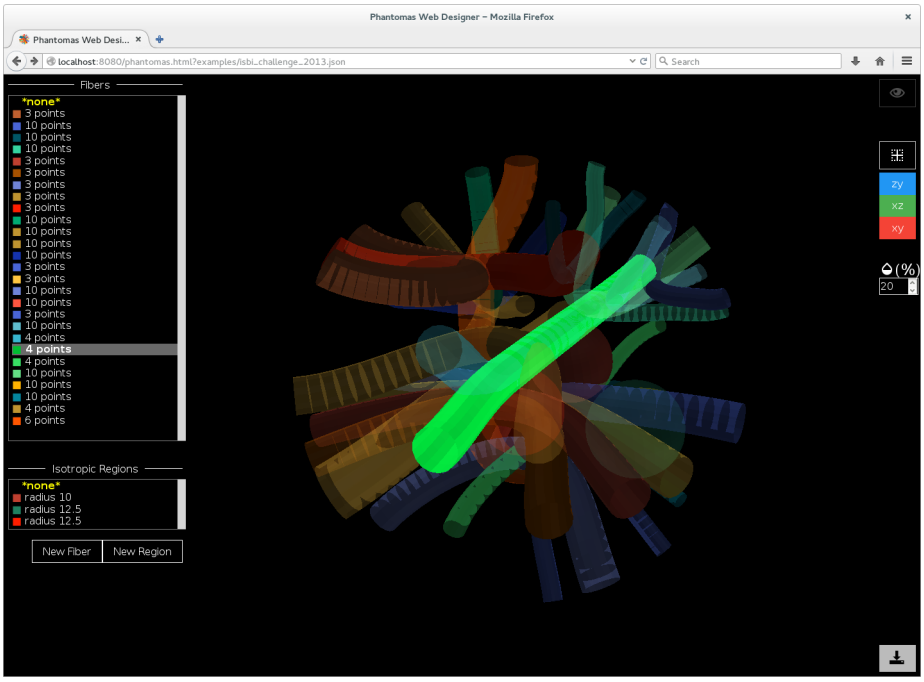



Figure 3.2: Screenshot of a fiber being placed the mouse over its entry in the element list and it being highlighted.

By clicking on an element, the application enters into edition mode. The user interface is then deployed and the lists are re-sized. A fiber and one of its control points being edited can be seen in figure 3.3. Note how the scene was easily moved by the user to place the editing element in the middle of the canvas.

As for the interactive edition, drag and drop controls were implemented. In figure 3.4 those may be seen in action. Note that the visual axis option is also enabled and that opacity is set to zero to ensure a more comfortable display.

By carefully watching the scene, one of the major problems faced when using *WebGL* and can also be noticed: the transparency. The way *WebGL* deals with the transparency needs an special attention *THREE.js* is unable to completely provide in this specific case.

During the coding process specific code had to be placed to fix this issue. Although it was almost solved, when rendering with transparency disk segments in tubular forms are unfortunately visible. Aliasing problems can also be appreciated when two meshes share a surface.

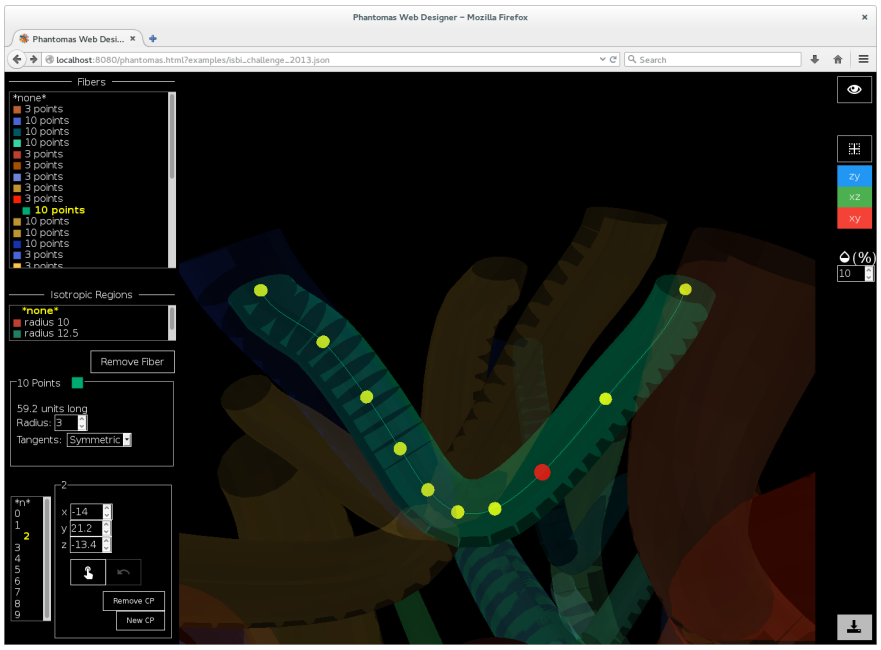



Figure 3.3: Fiber being edited. As edition mode was triggered, the user interface features more tools without taking more space. Note also how the point being edited is highlighted in the scene.

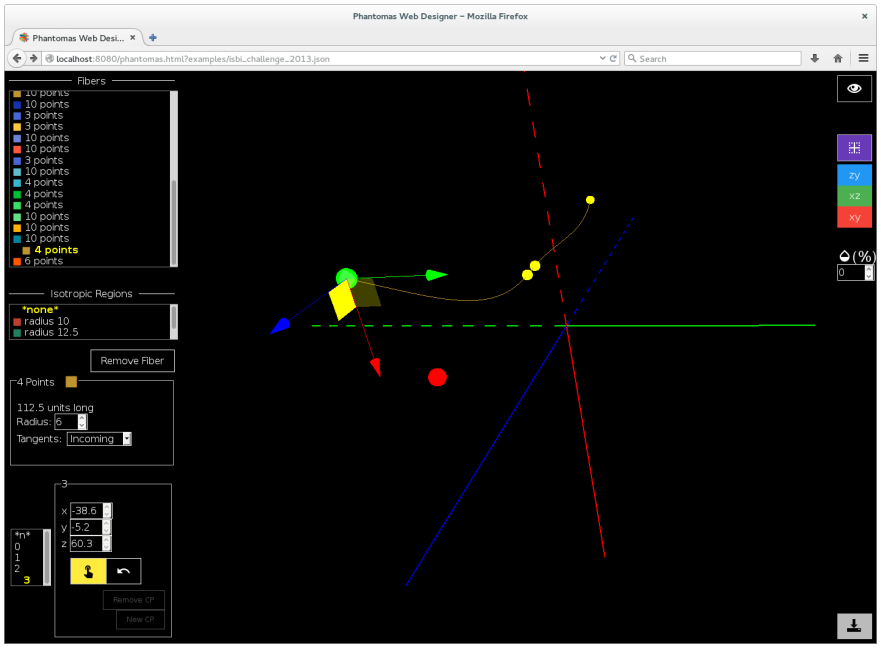



Figure 3.4: User using the interactive drag and drop tool to move a control point over a selected plane. The green point represents the actual while the red is the former. Undo button may be pressed at any time to recover the former position.

### *Phantomαs* Processing

*Phantomαs* compatibility was shown in the previous section by loading a phantom description specifically designed for *Phantomαs*.

In this section a phantom will be designed by using *Phantomαs* Web Designer and later, loaded and processed in *Phantomαs*.

For this demonstration a simple phantom example containing two fibers and a single isotropic region was created from scratch with *Phantomαs* Web Designer. Its representation is displayed in figure 3.5. Once its description was generated and downloaded, its JSON file was processed in *Phantomαs*. As introduced, *Phantomαs* includes a visualization script called `phantomas_view`, which renders a 3D phantom representation. A screenshot is available at figure 3.6. Note how the homogeneity is kept by representing it in a similar way, although `phantomas_view` is unable to read colors and exclusively sets random values.

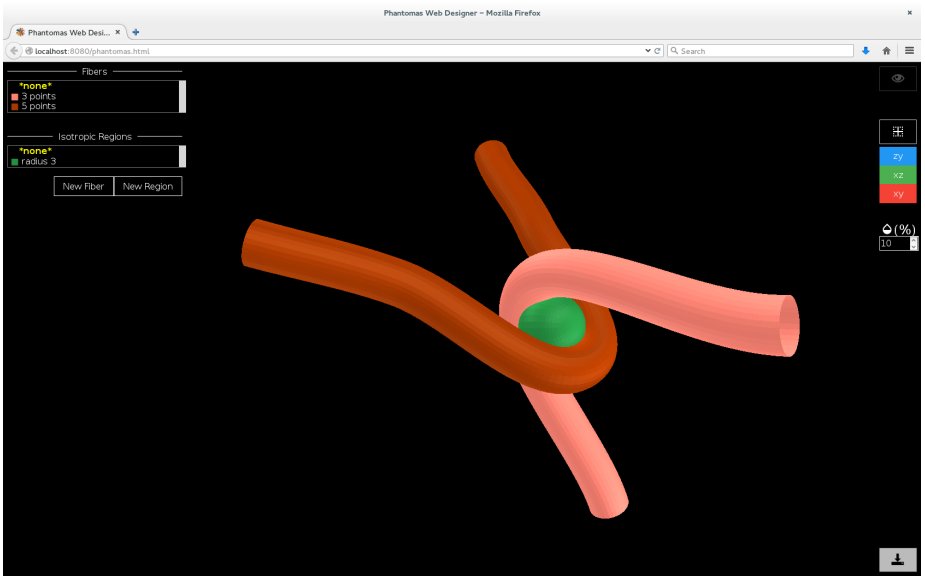



Figure 3.5: Phantom designed from scratch using *Phantomαs* Web Designer. Its description file was generated and later loaded with *Phantomαs*.

Once *Phantomαs* processes the phantom descriptions, a Diffusion Weighted Image (DWI) is given as output. In figure 3.7 both T1 and T2 DWI images are represented. From these images the tractography can be processed and its result compared to the ground truth. In figure 3.8 the representation of this phantom's tractogram is displayed by using the software *medInria* [5]. The two fibers in the phantom are easily recognizable in the predicted tracts.

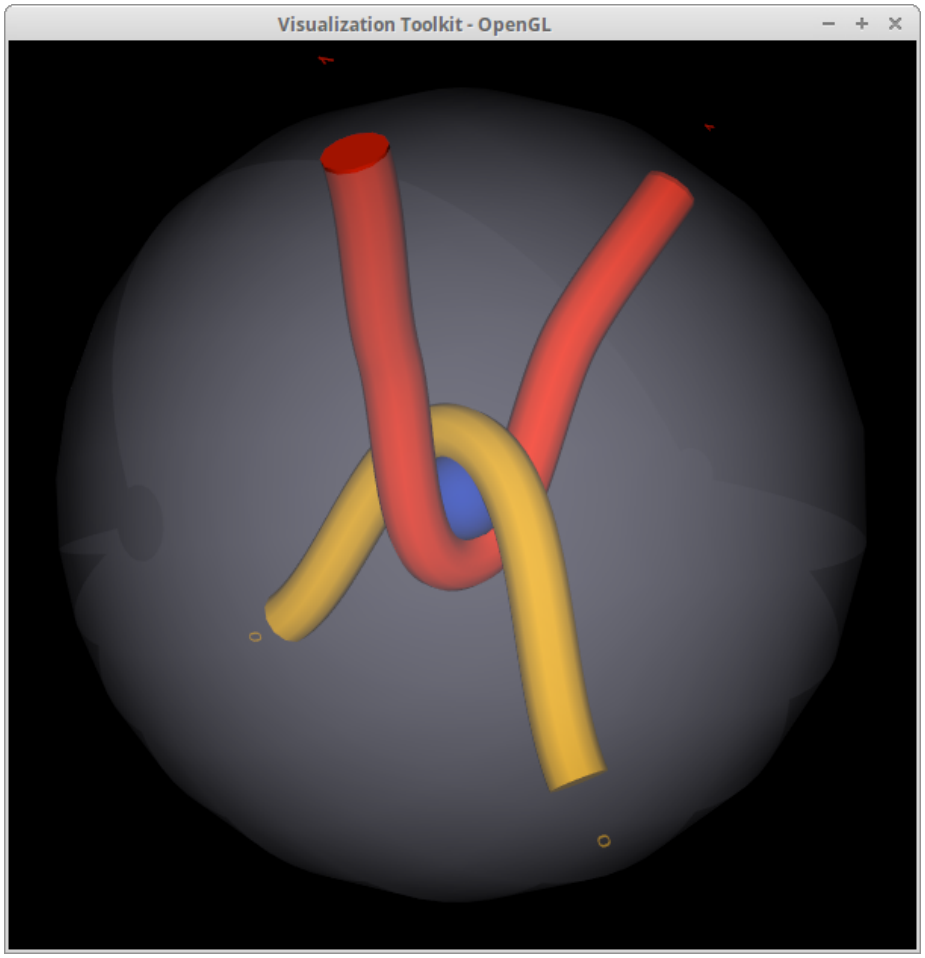
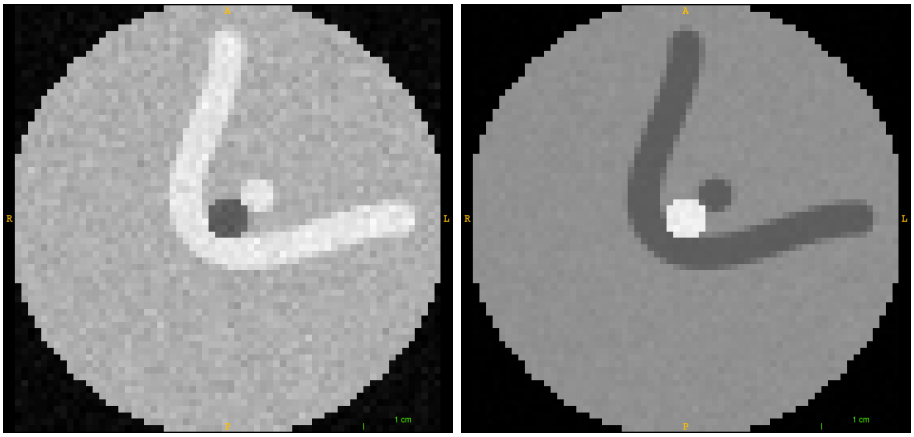



Figure 3.6: Designed phantom displayed with `phantomas_view` script, included in *Phantomαs*.



a b

Figure 3.7: Diffusion Weighted Image (DWI) output generated by *Phantomαs* after having processed the phantom's JSON file. Both show the middle plane cut. a. T1 DWI. b. T2 DWI.

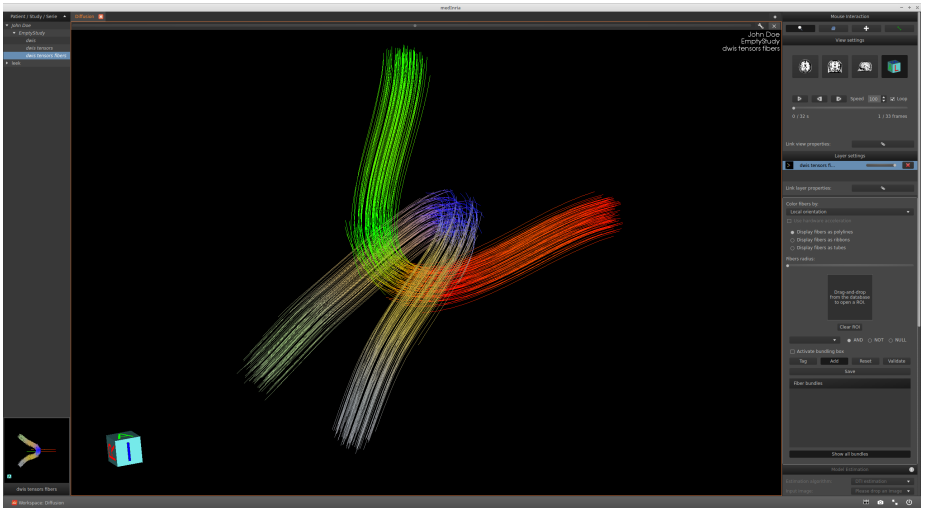


Figure 3.8: Processed tractography over the phantom, on *MedInria* [5].

# Chapter 4

# Environmental Impact

The impact in the environment the software may cause is considered negligible as long as it runs in the local client machine and does not require any server. Thus, we can consider its process unnoticeable among the other programs being simultaneously executed. Note that this is regarding to the actual version of *Phantomαs* Web Designer. The planned future development in *Phantomαs* involves a dedicated hosting server (see page 72).

The development impact can be estimated by summing up the consumption of the main tool, a desktop PC. $CO_2$ emissions were estimated in table 4.1 by considering an average consumption of 220 W[1].

| | |
|---|---|
| **Machine spend** | 220 W |
| **Working time** | 565 h |
| **$CO_2$ average emission in France** [36] | 52 g/kWh |
| **EMISSION ESTIMATE** | **6,464 g $CO_2$** |

Table 4.1: Development estimate $CO_2$ emissions.

[1] As estimated from the configuration used

# Chapter 5

# Budget

In this chapter we present a budget on the cost that developing a similar project would have on the market.

As it is a software project, it is understandable that the sum of engineering hours (table 5.1) places as the highest priced outlay, while the production means are easily redeemed. Establishment costs were also taken into account (table 5.2).

As every piece of the production was made with open-source software, this involves no costs and thus does not appear in the estimate.

In a professional project, several tests that did not actually take place in this project would have to be taken into account to ensure its quality.

| **Activity** | **Price** | **Amount** | **Cost (€)** |
|---|---|---|---|
| Project design | 35 €/h | 70 h | 2,450 € |
| Software development | 20 €/h | 425 h | 8,500 € |
| Software testing and bug fixing | 20 €/h | 70 h | 1,400 € |
| Documentation devising | 20 €/h | 35 h | 700 € |
| | | **TOTAL COST** | **13,050 €** |

Table 5.1: Staff costs.

| **Activity** | **Price** | **Cost (€)** |
|---|---|---|
| Electricity costs | 100 €/month | 500 € |
| Establishment costs | 500 €/month | 2,500 € |
| | **TOTAL COST** | **3,000 €** |

Table 5.2: Establishment and service costs.

Final cost: **16,050 €**

# Chapter 6

# Self Learning

As a Bachelor's Degree in Biomedical Engineering student, I chose this project for both its relationship with the discipline and my interest in software development.

By using the knowledge acquired during these four years studying at Universitat Politècnica de Catalunya I could develop this project. But many of the techniques used had to be self-learned during the development of the project.

Most of my self-learning is directly related to the technologies used for the development (page 11), and many resources have been used for learning their usage. Table 6.1 gathers the main technologies learned and the main resources used.

| **Technology** | **Main resources** |
|---|---|
| *JavaScript* | Book "JavaScript : the definitive guide" [18] |
| | W3Schools' *JavaScript* Reference [6] |
| | Mozilla Foundation's *JavaScript* Documentation [37] |
| *Three.js* | *Three.js*' official documentation [38] |
| *HTML* | W3Schools' *HTML* Reference [39] |
| | Mozilla Foundation's *HTML* Documentation [40] |
| *CSS* | W3Schools' *CSS* Reference [41] |
| | Mozilla Foundation's *CSS* Documentation [42] |
| *git* | Book: "Pro Git" [43], available online in *git*'s website [27] |

Table 6.1: Main self-learned technologies and their main resources studied.

# Chapter 7

# Conclusions

After having developed the entire application and tested its behavior, the result has been appreciated as satisfactory.

We consider that the application meets all of the initial requirements. In this chapter the final project will be compared with them (introduced in page 10) in order to prove they are all met.

**a) *Phantomαs* cross-compatibility**
*Phantomαs* Web Designer is able to load any JSON file yet used in *Phantomαs*, although at the moment this step is taken from the server side.

In the other direction, a JSON file with the exact same structure can be exported and loaded in *Phantomαs* for further processing.

We can consider this requirement is accomplished.

**b) Phantom display and WYSIWYG edition**
The phantom canvas display takes the main part of the page, gathering the most attention of the user at all time. It is completely interactive and the user is able to freely change the view with simple mouse gestures or through specific controls placed at the right panel of the page.

In addition, any edition step made is shown in the visual display continuously, making the display the main input of information to the user. The result obtained from the current edition is always visible, allowing the user to see how the exported phantom will be and thus, implementing the WYSIWYG (*what-you-see-is-what-you-get*) edition.

We can consider both of these requirements are accomplished.

**c) Wide edition control**
Editing all those properties relevant to the geometry processing, was the core target of *Phantomαs* Web Designer.

The final application allows all of these modifications to be taken on element of a phantom. For the fiber bundles, this involves:

- Adding new fiber bundles
- Removing fiber elements at the phantom
- Changing the tangent computation method
- Specifying a fiber radius
- Editing control points:
  - Adding new control points
  - Removing existent control points
  - Changing the position of a control point

On top, the position of a control point can be edited either by selecting the exact point in the area, or by drag and dropping it in the scene. Undoing changes is also available for this tool.

As for the isotropic regions, the edition tools allow:

- Adding new isotropic regions
- Removing isotropic regions from the phantom
- Editing their radius
- Specifying their center position

We have shown that the user is able to edit all those characteristics in a phantom, so we may consider this requirement is fairly accomplished. Adding more interactive tools to ease the user experience could be considered as a feature extension.

**d) Versatility**
Being the web environment the chosen for the development, the application may be run in most desktop environments. This makes *Phantomαs* Web Designer an universal application and sets it as a good base for the future development of *Phantomαs*. Note that *WebGL* is also implemented in last mobile devices models, which would also able to execute the code, although the application is not feasible as its design did not target them at all.

As for the requirements needed for running the software, those are satisfied by using an average machine capable of running a modern web browser which supports *WebGL* technologies. As of May 2017, over 92% of active web desktop clients in the world were using such software, whereas in France almost 96% were [44]. this along with the fact that for using a web application any kind of installation is needed makes *Phantomαs* Web Designer a very versatile application.

Expected user (see page 15) includes researchers who have as main working tool a modern desktop environment. In this way, we can ensure this requirement is accomplished.

After testing the application and experiencing the edition, we can also conclude that our web environment is capable of perfectly hosting all the requirements *Phantomαs* Web Designer needs.

The web environment is currently hosting applications that in the past were never set to be executed in a browser. Many of the tools used during the development (see page 11) are also based on web technologies. This fact proves this environment is becoming stable and that our application will stay feasible in end-clients' computers for long term.

# Chapter 8

# Future Extensions

*Phantomαs* Web Designer only concerns the design of phantoms and generating its description for *Phantomαs*, but it is just part of a project that will be developed in the future.

Although *Phantomαs* was perfectly functional, the amount of dependencies and its lack of user interface was making it a tedious tool for some users. Having evaluated the situation, a new approach was set by *Phantomαs*' designers: developing an entirely functional web-based version of the library.

*Phantomαs* Web designer is the first piece of this web environment. When its development is finished it will allow the users to edit their phantoms, set the MRI options and download their processed DWI images. This process is schematized in figure 8.1.

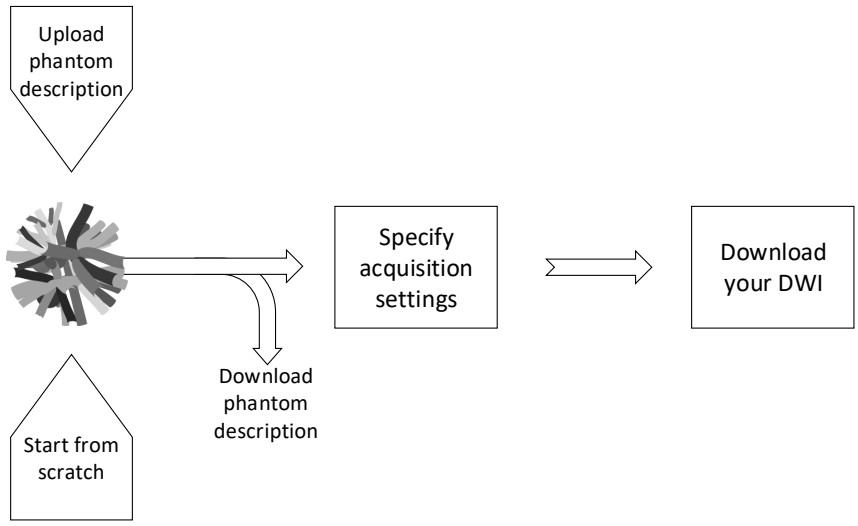



Figure 8.1: Schema of what the future *Phantomαs* web environment is meant to be.

# Bibliography


[1] E. Caruyer, L. Bloy, B. Tunç, J. Lecoeur, V. Shankar, and R. Verma, "A comparative study of 16 tractography algorithms for the corticospinal tract: reproducibility and subject-specificity," in *ISMRM*, (Milan, Italy), May 2014.

[2] E. Caruyer, A. Daducci, *et al.*, "Phantomas: a flexible software library to simulate diffusion MR phantoms," in *ISMRM*, (Milan, Italy), May 2014.

[3] E. Caruyer, "Phantomas 0.1 documentation." `http://www.emmanuelcaruyer.com/phantomas/`, 2017-05-14.

[4] E. Caruyer, "ecaruyer/phantomas: Create realistic digital phantoms in diffusion MRI.." `http://github.com/ecaruyer/phantomas`, 2017-05-14.

[5] VisAGeS Research group, "medInria." `https://med.inria.fr/`, 2017-04-22.

[6] W3C, World Wide Web Consortium, "JavaScript and HTML DOM Reference." `http://www.w3schools.com/jsref`, 2017-05-16.

[7] "Inria, un établissement public de recherche dédié aux sciences du numérique - Inria." `http://www.inria.fr/institut/inria-en-bref/inria-en-quelques-mots`, 2017-04-18.

[8] "www.irisa.fr | Institut de Recherche en Informatique et Systèmes Aléatoires." `http://www.irisa.fr/`, 2017-04-18.

[9] "VisAGeS » Vision, Action and information manaGement System in health." `http://team.inria.fr/visages/`, 2017-04-19.

[10] E. Caruyer, "Emmanuel Caruyer homepage — CNRS Researcher, IRISA (UMR 6074), VisAGeS Research group, Rennes FR-35042.." `http://emmanuelcaruyer.com/`, 2017-05-05.

[11] D. G. Taylor and M. C. Bushell, "PRELIMINARY COMMUNICATION: The spatial mapping of translational diffusion coefficients by the NMR imaging technique," *Physics in Medicine and Biology*, vol. 30, pp. 345–349, Apr. 1985.

[12] V. Wedeen, R. Wang, *et al.*, "Diffusion spectrum magnetic resonance imaging (dsi) tractography of crossing fibers," *NeuroImage*, vol. 41, no. 4, pp. 1267 – 1277, 2008.

[13] K. Maier-Hein, P. Neher, *et al.*, "Tractography-based connectomes are dominated by false-positive connections," *bioRxiv*, 2016.

[14] E. Caruyer, "Phantomas - Simulate realistic diffusion MRI Phantom - Emmanuel Caruyer homepage — CNRS Researcher, IRISA (UMR 6074), VisAGeS Research group, Rennes FR-35042.." `http://emmanuelcaruyer.com/phantomas.php`, 2017-04-20.

[15] "Welcome to Python.org." `http://www.python.org/`, 2017-05-05.

[16] "HARDI reconstruction challenge 2013 - HARDI reconstruction challenge 2013." `http://hardi.epfl.ch/static/events/2013_ISBI/`, 2017-05-05.

[17] Red Hat, Inc., "Fedora." `http://getfedora.org/`, 2017-05-05.

[18] D. Flanagan, *JavaScript : the definitive guide*. Beijing Sebastopol, CA: O'Reilly, 2011.

[19] W3C, World Wide Web Consortium, "Open Web Platform Milestone Achieved with HTML5 Recommendation." `http://www.w3.org/2014/10/html5-rec.html.en`, 2017-05-04.

[20] "three.js - Javascript 3D library." `http://threejs.org/`, 2017-05-05.

[21] Mozilla Foundation, "Blocklisting/Blocked Graphics Drivers - MozillaWiki." `http://wiki.mozilla.org/Blocklisting/Blocked_Graphics_Drivers`, 2017-05-04.

[22] Mozilla Foundation, "Download Firefox — Free Web Browser — Mozilla." `http://www.mozilla.org/en-US/firefox/new/`, 2017-05-05.

[23] "Normalize.css: Make browsers render all elements more consistently.." `http://necolas.github.io/normalize.css/`, 2017-05-05.

[24] "Node.js." `http://nodejs.org/`, 2017-05-05.

[25] "Use JSDoc: Index." `http://usejsdoc.org/`, 2017-05-05.

[26] "reStructuredText Markup Specification." `http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html`, 2017-05-05.

[27] "Git." `http://git-scm.com/`, 2017-05-05.

[28] GitHub, Inc., "GitHub." `http://github.com/`, 2017-05-05.

[29] GitHub, Inc., "Atom." `http://atom.io/`, 2017-05-05.

[30] "mrdoob/three.js: JavaScript 3D library.." `http://github.com/mrdoob/three.js/`, 2017-05-15.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[32] User "volzo", "javascript - Download JSON object as a file from browser - Stack Overflow." `http://stackoverflow.com/questions/19721439/download-json-object-as-a-file-from-browser`, 2017-05-17.

[33] W3C, World Wide Web Consortium, "W3.CSS Home." http://www.w3schools.com/w3css, 2017-05-17.

[34] Google Inc., "Material Icons — Material Design." http://material.io/icons/, 2017-05-17.

[35] G. González Vela and E. Caruyer, "ecaruyer/phantomas-web: Web interface for Phantomas.." http://github.com/ecaruyer/phantomas-web, 2017-05-14.

[36] RTE France, "eco2mix co2 en | RTE France." http://www.rte-france.com/en/eco2mix/eco2mix-co2-en, 2017-05-20.

[37] Mozilla Foundation, "JavaScript | MDN." https://developer.mozilla.org/en-US/docs/Web/JavaScript, 2017-05-20.

[38] "three.js docs." https://threejs.org/docs/, 2017-05-20.

[39] W3C, World Wide Web Consortium, "HTML Reference." https://www.w3schools.com/tags/default.asp, 2017-05-16.

[40] Mozilla Foundation, "HTML | MDN." https://developer.mozilla.org/en-US/docs/Web/HTML, 2017-05-20.

[41] W3C, World Wide Web Consortium, "CSS Reference." https://www.w3schools.com/cssref/default.asp, 2017-05-16.

[42] Mozilla Foundation, "CSS | MDN." https://developer.mozilla.org/en-US/docs/Web/CSS, 2017-05-20.

[43] S. Chacon, *Pro Git*. Berkeley, CA New York, NY: Apress,Distributed to the Book trade worldwide by Spring Science+Business Media, 2014.

[44] A. Deveria and L. Schoors, "Can I Use." http://caniuse.com/, 2017-05-20.