Web-based visualisation of MRI cardiac images

Torres Bisbal, Melanie

Year 2014-2015

Directors: Alun Evans, Gemma Piella and Oscar Cámara

Simultaneous studies program between degrees:
Bachelor’s degree in Audiovisual Systems Engineering and Bachelor’s degree in Computer Sciences
Web-based visualisation of MRI cardiac images

Melanie Torres Bisbal

BACHELOR FINAL PROJECT

Simultaneous studies program between degrees:

Bachelor's degree in Audiovisual Systems Engineering and Bachelor's degree in Computer Sciences

ESCOLA SUPERIOR POLITÈCNICA UPF

Department of Information and Communication Technologies (DTIC)

YEAR 2015

DIRECTOR/S OF THE BFP:

Alun Evans (GTI),

Gemma Piella (SimbiosysGem),

Oscar Cámara (PhySense)
Dedicated to my mother for all the encouraging words that she has said to me over the years.
Dedicated to my father because he is the reason for doing a project related to health care.
Dedicated to my sister and all my sailor friends.
Dedicated to min älskling.
Acknowledgements

First of all I would like to thank Alun Evans, Gemma Piella and Oscar Cámara for being my supervisors and guiding me during the project. I want to thank all the GTI, especially the computer graphics group and Josep Blat for giving me support from a research department and giving me the opportunity to make a project that motivates me with them. I would like to thank the Inria Sophia-Antipolis team for giving me the possibility to work with a foreign research group.

I also want to thank all my closest friends that were suffering all my complainings and helping me in all they could. I am grateful to all my classmates that were with me during these years and made all those hours together funny and entertaining, especially Roser and Sandra.

I would like to thank my mom and my sister for supporting and helping me during these years despite the difficulties. Finally, my sincere gratitude to Andreas for all the support and trust, the thoughts that I would be able to make the project from the first moment without hesitation and all his encouraging words.
Abstract

Recent years have seen an explosion in the popularity and availability of web-based applications that allow direct creation and editing of content (such as documents, photos and video) within the browser. One of the reasons for this popularity is that such tools offer many features that are difficult or impossible to achieve with their offline equivalents, such as collaborative and remote working, easy sharing of content, remote visualisation, annotations, etc. The healthcare sector is not an exception and there is an increasing number of visualisation and image processing platforms starting to port their products towards web-based solutions that allow features such as remote second medical opinions. Nevertheless, most of these advances are still in their infancy and they have not really been translated into a clinical environment.

MedInria (http://med.inria.fr/) is a multi-platform medical image processing and visualisation Open-Source software, which does not yet explored the web as a tool for its purpose; yet there is a strong incentive to bring the benefits of web applications to MedInria, as it could potentially help doctors with their daily tasks, such as diagnosis, consultation, interventions, etc.

The main objective of this project is to create a Web-based visualization based on the MedInria software, allowing the sharing of content between remote users, permitting them to give feedback and to discuss 3D multimodal data. The main research challenges that have been tackled in this project include rendering 3D medical meshes in a web context, user interactivity with the medical data (e.g. annotations, joint visualization of images and meshes, etc.) and the connection between the web viewer and the stand-alone MedInria application. The developed application has been tested on a pilot case to visualize and interactively process MRI data of the heart as well as to integrate it with electro-anatomical mapping data of the same patient. The application was tested by several healthcare professionals. Very positive feedback and potential improvements for a clinical translation of the developed tool have been gathered in the evaluation process.
Resumen

En los últimos años ha habido una explosión en la popularidad y disponibilidad de las aplicaciones basadas en la web que permiten la creación y edición de contenido directo y la edición de contenidos (como documentos, fotos, vídeos) en el navegador. Una de las razones de esta popularidad es que estas herramientas ofrecen muchas características que son difíciles o imposibles de alcanzar con sus equivalentes sin conexión, como el trabajo colaborativo y remoto, compartir contenido fácilmente, visualización remota, anotaciones, etc. El sector de la salud no es una excepción y hay un número creciente de plataformas de visualización y procesamiento de imágenes a partir de portar sus productos hacia soluciones basadas en la web. Sin embargo, la mayoría de estos avances están todavía en una fase muy temprana y realmente no se han implementado en un entorno clínico.

MedInria (http://med.inria.fr/) es un software Open-Source multi-plataforma de procesamiento y visualización de imágenes médicas, que todavía no ha explorado la web como una herramienta; sin embargo, hay un fuerte interés para que los beneficios de las aplicaciones de la web se utilicen medInria, ya que podría ayudar a los médicos con sus tareas diarias, como el diagnóstico, consulta, intervenciones, etc.

El objetivo principal de este proyecto es crear una visualización basada en la Web tomando como base el software medInria, permitiendo el intercambio de contenidos entre usuarios remotos, lo que les puede permitir dar información y discutir datos multimodales 3D. Los principales retos que hemos afrontado en este proyecto incluyen renderizar las mallas médicas 3D en un contexto web, la interacción de los usuarios con los datos médicos (como las anotaciones, visualización simultánea de las imágenes y mallas, etc.) y la conexión entre el usuario de la web y la aplicación stand-alone medInria. La aplicación desarrollada ha sido puesta a prueba en un caso piloto para visualizar y procesar de forma interactiva los datos de una resonancia magnética del corazón, así como para integrarlo con los datos de mapas electro-anatómicos del mismo paciente. La aplicación fue probada por varios profesionales de la salud. En tal proceso de evaluación se recibió un feedback muy positivo y propuestas para potenciales mejoras en caso de que esta herramienta se trasladara a un ámbito clínico.
# Index

Abstract vii
List of figures xii
List of tables xiv

## 1. Introduction 1
1.1. Personal Context 3
1.2. Motivation 4
1.3. Objectives 5

## 2. Background 9
2.1. Medical Informatics 9
2.2. Magnetic Resonance Imaging 10
2.3. The heart 10
2.4. Medical formats and standards 13
2.5. Applications of the visualisation in medical imaging 16
  2.5.1. Offline 16
  2.5.2. Online 20
2.6. Enabling Technologies 23

## 3. Methodology 31
3.1. Design 33
  3.1.1. Use cases 33
  3.1.2. Requirements of the application 34
  3.1.3. Study of how to address the design 34
  3.1.4. Decision about the application 35
3.2. Preparatory work 35
  3.2.1. Getting started with Javascript 35
  3.2.2. Visualizing data 37
  3.2.3. Visualising VTK data in the library XTK 40
  3.2.4. Visualising the MRI in the application 42
3.3. Implementation 44
  3.3.1. Summary of modifications on the XTK library 44
  3.3.2. Changing shader parsing method in XTK 46
  3.3.3. Creating the binary mask 46
  3.3.4. Adding a second render 49
  3.3.5. Process of finding the maximum and minimum 50
  3.3.6. Paint over the magnetic resonance using a mask 51
  3.3.7. Both renders are rotating and interacting at the same time 53
  3.3.8. Annotation on the application 54
  3.3.9. Connection with medInria 58
  3.3.10. Configuration file 59

## 4. Results 61
4.1. Final application 61
4.2. Feedback 64

## 5. Conclusions 67
5.1. Evaluation 67
5.2. Future work 68
5.3. Personal evaluation 69

Bibliography 73

Appendix 77
  Figure links 77
List of figures

Figure 1.1. Synthesis of the digital technologies involved with health .............................................2
Figure 2.1. The human heart ...........................................................................................................11
Figure 2.2. Electrical system of the heart .......................................................................................12
Figure 2.3. Difference between a normal heart and one with atrial fibrillation .........................13
Figure 2.4. Visualization in medInria ............................................................................................17
Figure 2.5. Registration between MRI and MDCT data into a fused image series .....................17
Figure 2.6. Multimodal three-dimensional model .........................................................................18
Figure 2.7. At the application framework, we can see the data and mask that we are using ..........18
Figure 2.8. View of the 3D Slicer application .................................................................................19
Figure 2.9. Example of visualisation of data ..................................................................................20
Figure 2.10. Example using Slice:Drop .........................................................................................21
Figure 2.11. Papaya viewer ...........................................................................................................21
Figure 2.12. Example of the ScanView application .......................................................................22
Figure 2.13. CPU vs GPU architecture .........................................................................................25
Figure 2.14. Summary of web technologies to paint .....................................................................26
Figure 2.15. Diagram of Graphics Pipeline ....................................................................................27
Figure 3.1. Steps for initialize the application ..................................................................................32
Figure 3.2. Ways to interact with the application and steps that it takes .....................................33
Figure 3.3. Example of the communication between the client and the server ...........................36
Figure 3.4. Examples of different application model .......................................................................36
Figure 3.5. Visualisation of the MRI of a patient with an atrial fibrillation disease in MITK .........37
Figure 3.6. Cardiac model of a patient with an atrial fibrillation disease .................................38
Figure 3.7. Selection of some cells of the right ventricle .............................................................39
Figure 3.8. Zoom on the selection inspector ....................................................................................39
Figure 3.9. Zoom on the filters .......................................................................................................40
Figure 3.10. Screenshot of the final result with only the left ventricle .........................................40
Figure 3.11. CARTO data without any lookup table .................................................................41
Figure 3.12. CARTO data with BIP lookup table ..........................................................................42
Figure 3.13. CARTO data with PI lookup table ............................................................................42
Figure 3.14. MRI on the MTIK software .........................................................................................43
Figure 3.15. MRI on the XTK application .......................................................................................44
Figure 3.16. XTK diagram of the main classes .............................................................................45
Figure 3.17. Screenshot of the data displayed of the MRI in MITK .............................................47
Figure 3.18. Example of what does the vtkPolyDataToImageStencil filter ..............................48
Figure 3.19. Screenshot of all the renders added into the application .........................................50
Figure 3.20. Formula for consider a region unhealthy .................................................................51
Figure 3.21. Pipeline of the GPU ...................................................................................................52
Figure 3.22. Example of the unhealthy zone and the contours .....................................................53
Figure 3.23. Data showed in the application ..................................................................................53
Figure 3.24. Visualisation of the three planes ................................................................................55
Figure 3.25. Example of the line of annotation in the application ................................................56
Figure 3.26. Window to enter the text into the annotation ............................................................56
Figure 3.27. Annotation in the application ....................................................................................57
Figure 4.1. Final application ..........................................................................................................61
Figure 4.2. MRI data .....................................................................................................................62
Figure 4.3. 3D model of the left ventricle ................................................................. 62
Figure 4.4. Planes of the slices ................................................................................... 62
Figure 4.5. Controls of the application ........................................................................ 63
List of tables

Table 2.1. Example of NRRD file.................................................................15
Table 2.2. Summary of XTK interaction......................................................29
Table 3.1. Example of the configuration file ..............................................59
Table 4.1. Summary of data in the application ..........................................62
1. Introduction

Cardiovascular diseases are one of the most important problems in developed countries. In 2005, the total number of cardiovascular disease deaths had increased globally to 17.5 million from 14.4 million in 1990. It is estimated that by 2030 this type of disease will account for more deaths than infectious, maternal and perinatal conditions, and nutritional disorders combined (Countries, 2010).

Cardiac arrhythmias are one of these cardiovascular problems. They occur when the electrical impulses that coordinate the heartbeats do not work properly, causing the heart to beat too slow, too fast or irregularly. In each heartbeat an electrical signal fires from the top to the bottom of the heart, which makes that the heart contract and pump blood (National Heart Lung and Blood Institute, 2011). One type of arrhythmia in particular is called atrial fibrillation, which makes the signal travel in a fast and disorganized way, this causes the heart’s two upper chambers (called the atria) to fibrillate, due to an area (structure or tissue) of the heart which is problematic (National Heart Lung and Blood Institute, 2014). Most treatments for this problem involve the ablation, removing or melting away of the problematic section.

The main objective of this project is to create a Web-based visualization based on the medInria software, allowing the sharing of content between remote users, permitting them to give feedback and to discuss 3D multimodal data. It has been developed in the area of online medical applications. It was a collaboration with three groups from the UPF (GTI, PhySense and Simbiosys) and Inria Sophia-Antipolis (French institution that developed medInria within the Asclepios research project).

Thus, this project intends to contribute to healthcare systems in general, and in particular to the problem commented above, regarding cardiac arrhythmias. Currently, Information and Communication technologies (ICT) are commonly integrated in healthcare systems, providing tools to clinicians that help them in daily processes like diagnosis, interventions, patient selection and stratification, hospital management, acquisition, storage and retrieval of information, among other applications.
Such tools are currently improving healthcare systems in the following aspects (Chaudhry et al, 2006):

- Improve healthcare quality
- Improve healthcare productivity
- Prevent human errors
- Reduce healthcare costs
- Reduce time in the process that can be automatised like paperwork
- Extend the real time communications to the healthcare sector and bring all that opportunities there
- Expand the access of the care

Currently, information technology (IT) is quite integrated in many areas such as clinical, consumer health, dental, mental health, nursing, pharmacy, primary care, public health, telemedicine and mobile computing, veterinary and also in clinical research in order to help to understand the physiology of our body, simulate it and try to understand it in a better way, and make conclusions for treatment strategies from data analysis. We can observe a visual synthesis of the digital technologies involved with health in the Venn Diagram of figure 1.1.

![Venn Diagram of figure 1.1. Synthesis of the digital technologies involved with health [1]](image-url)
Nonetheless, IT in health care could be more exploited and integrated, following the example of the collaborative work applications via the web (e.g., Google Drive) but oriented to the medical environment (American Dental Education Association, 2012).

In medical informatics there is a need for visualising and pre-processing medical images since it is a very powerful source of medical information. Several scanner company-based (e.g. Philips, General Electric, Siemens) and academic research groups have developed advanced image processing and visualization tools providing clinician-friendly stand-alone interfaces allowing healthcare professionals to process their patient data. Some examples of them are Slicer3D, Seg3D, ITK-Snap, OsiriX, medInria and many more. The one that we are going to focus is medInria. MedInria (http://med.inria.fr/) is a multi-platform medical image processing and visualisation Open-Source software, which has not yet explored the web as a tool for its purpose; yet there is a strong incentive to bring the benefits of web applications to medInria, as it could potentially help doctors with their daily tasks, such as diagnosis, consultation, interventions, etc.

1.1. Personal Context

This project represents the joint deliverable of two degrees, which I am currently studying: the Degree in Audiovisual Systems Engineering and Degree in Computer Sciences. Before deciding the theme of the project I talked with Oscar Cámara and Gemma Piella to let them know that I was interested in doing my final projects with them, addressed in the field that they are associated, informatics related with medicine. In my fourth year of university, I started a scholarship in the group of interactive technologies (GTI) in the computer graphics section. When it was the time to start thinking about the final project, we considered the possibility of doing a collaborative project with the groups of Gemma and Oscar, and the GTI.

The GTI has a background in working in remote environments in web and computer graphics so we thought about taking advantage of that part and making a web based visualisation of medical data and connect it with an existing desktop application for visualising medical data like the one developed in Inria (medInria).
Therefore, this final degree project has taken place with the help and collaboration with three research groups from the UPF (GTI, PhySense and Simbiosys) and Inria Sophia-Antipolis (French institution that developed medInria within the Asclepios research project)

1.2. Motivation

The motivation of this application is to provide new functionalities that have not been covered before. One of those functionalities (which are detailed further in the objectives section below) is rendering 3D data in a web context. The reason to make it in the web is to sideline issues regarding multiplatform about programming, and reducing the need to install programs, as we can use the common web-browser context.

Currently, clinicians need much more than just observing the data from the patients, they need different ways to interact with it. That is why it is so important to have some level of interactivity like joint visualization of images and meshes, being able to change thresholds for observe different areas of the data in various modes.

A step further thinking about interactivity is interactivity with the data and with other persons, thinking about that is where the concept of “second opinion” may seem useful. It has long been seen as a valid approach in medical diagnosis, yet it is not easy to access relevant experts to ask for their opinion. The idea therefore is to remove geographic barriers and be able to have different opinions from the best specialists in the area independently of geographical distance. Many hospitals, research centers or even universities in the United States are offering these kinds of services. A few examples are the University of Pennsylvania Health System, Mount Sinai Hospital, UC San Diego Health System, University of Colorado Health, Johns Hopkins Medicine and many more.

There is also a new concept of e-Patient where the patient also participates in his or her medical care. They are empowered and equally positioned with the physicians involved in their health (Ostherr et al, 2015). All of this has been facilitated by access to information on the internet. So an application for sharing data can be addressed in a
different ways, not only clinicians asking for second opinions, also patients asking for them.

We can see a future in the medical age enabled by technologies, also redefining what it means to be a doctor. We are seeing a future that is networked and we have to start to give tools to it.

1.3. Objectives

The main objective of this Bachelor’s Final Project (BFP) is to develop a web-based application for visualization based on the medInria software, allowing the sharing of content between remote users, permitting them to give feedback and to discuss 3D multimodal data. The project’s aim is to create a system for the simplest case with a single data source, with the possibility of extending the work if it proved to be successful.

In order to achieve this objective, it is necessary to export the data and the configurations displayed in medInria and transfer that information into a web application. The application has to know how to parse this information and display what medInria shows in the stand-alone application in the most accurate way possible and with the same parameters. It should also include a feature to add annotations to the file. These annotations are useful to exchange information about particular parts of the image where special attention needs to be paid. Thus, they would make easier to obtain second opinions for the diagnosis of heart disease, regardless of the geographic barriers.

The application has a few standard and interactive image processing tools such as segmentation by thresholds. This tool is important, because it can provide important information between different users. Another aim of this application is to promote collaboration using a web-based visualisation interface, which is accessible on any platform with a modern web-browser and from any location.

This application is not a substitute for the available stand-alone applications, but it provides new functionalities such as rendering 3D medical meshes in a web context,
user interactivity with the medical data (e.g. annotations, joint visualization of images and meshes, etc.).
2. Background

Cardiovascular diseases are one of the most important problems in the developed countries and Medical informatics can help in their early detection and treatment.

2.1. Medical Informatics

Medical informatics are the part of science that is related to the use of informatics to acquire, store, analyze, communicate and display medical information and the ways to facilitate the understanding of the information to improve the accuracy, timeliness and reliability of decision-making (Homer et al, 2015). Currently it is hard to think about medicine without all the benefits that informatics are able to give us.

Within the medical informatics there is a field called medical imaging, which includes the techniques and processes to create a representation of the interior of the body to visualise it.

Nowadays we are able to visualise body parts, tissues and organs. It is used in clinical diagnosis and treatment, to help in interventions and other fields that could take advantage of it.

The most used scanning devices are X-ray Computerized Tomography (CT), Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET).

CT is a method that combines multiplies X-Ray projections. MRI use scanners with magnetic fields and radio waves to create detailed information about the structure and function of the human body (see section 2.3 for more details). PET is a nuclear imaging technique that gives information about how tissues and organs are operating.

In case a volume is involved the usual structure of the data are plane X (Sagittal), Y (Axial) and Z (Coronal) images.

To get an idea of how important medical imaging is nowadays and the impact that it has in research, we can take a look at the data and statistics available on the web. If we access to the web IEEE Xplore (an important scholarly research database) and look for topics related to medical imaging we can see that it has a section related to it.
2.2. Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) is an imaging modality that provides detailed information about the structure and function of the human body, it allows the visualisation of the diseased and damaged tissue.

It is based on a phenomenon known as Nuclear Magnetic Resonance that was first described and measured in 1938 by Isidor Rabi. Since those years to nowadays MRI has become an important tool for the diagnostic with imaging. In our case, Cardiologists can get images of the heart to help in the diagnosis (The University of Queensland, 2014).

It is one of the most computerized imaging techniques used in any scale, even in hospitals in smaller cities and towns. One of the reasons of that is that magnetic resonance could give us information that any other imaging method could not (Bankman et al, 2002).

2.3. The heart

A healthy heart consists of four chambers (two atria and two ventricles). The atria are positioned above the ventricles, and their function is to provide blood to the ventricles. The function of the ventricles consists in delivering blood to the body and the lungs. In a proper functioning, blood from the venous system is acquired from the right atrium and then pumped into the right ventricle. The right ventricle sends blood by the pulmonary arteries to the lungs and travels to the left atrium and into the left ventricle. The left ventricle is the most important chamber of the heart because it has to generate the pressure to pump blood to the body (Bankman et al, 2002). For this reason we focus on the left ventricle, considering it the main region of interest. We can see the chambers of the heart in figure 2.1.
As said before, there are four heart chambers that are showed in the figure 2.1, the right and left atrium and the right and the left ventricle where the blood is going to flow of all of them.

There is a special kind of cells into the sinoatrial node (SA) that can depolarize by themselves without the help of any other cell: that is called automaticity. That means that they are able to squeeze out that chamber of the heart. We can observe where the SA is in figure 2.2.

After that, those cells are going to start sending waves of depolarization to the surrounding cells. Even though, there is a highway between the right atrium and the left atrium which is called Bachmann’s Bundle, that is going to take the depolarization into the other chamber in a coordinated way. Besides, there is another type of tissue that takes the electrical signal to another node, called atrioventricular node (AV). In a normal heart this node is the only connection between the atrium and ventricles. The connections between the SA and AV are called Internodal Tract.

There is a tiny delay of 0.1 seconds between the atria and the ventricles due to the AV node. The delay allows the blood to move in a coordinated way through the heart.

After that, the signal goes into his bundle, an area between the AV node and the separation of the two bundle branches (right and left). Then it splits into the two bundle branches and then, all the fibers split even more in both sides. These are the conduction
pathways. From this point on, the electrical signal can dash out in all directions and all the muscles cells are involved in that process. (Khan Academy Medicine, 2012).

![Figure 2.2. Electrical system of the heart [3]](image)

Atrial fibrillation is a heart arrhythmia (Afib), one of the supraventricular tachycardias. In a normal heart the electrical system acts just the way we explained before. However, in an Afib the electrical waves are in disarray. They are erratic and they go everywhere, and there is not going to be a unified contraction of the atria: instead, the atria will start spasming. In the figure 2.3 we can observe this kind of behavior and also an electrocardiography (ECG) in which we can appreciate the difference between both of them. In the atrial fibrillation the beats are irregular which means that the distance between the peak and the next one is different in every interval (Khan Academy Medicine, 2014).
Figure 2.3. Difference between a normal heart and one with atrial fibrillation [4]

The main data that we have is an MRI of the thorax where we are able to visualise the heart and all the organs and tissues that we have in the thorax. This data shows a non-healthy part of the heart, and that region is called *scar*. The scar can produce problems such as arrhythmias that we commented before. The usual treatment is performing a cardiac ablation. The software that we are developing will help in this type of data, so in this project we are only going to work with this.

### 2.4. Medical formats and standards

The first time that we observed the magnetic resonances they were in a DICOM file. DICOM is the acronym of Digital Imaging and Communications in Medicine and defines a protocol of interchanging data between two entities. It works above the TCP/IP protocol (this enables to communicate over the internet), digital image format and file structure. It is a standard to specify all the elements required for making a system that is quite automatic and it is able to operate between biomedical imaging
computers. There is a method that allows us to obtain objects by web access, called WADO (Web Access to DICOM Objects).

At the present time, this standard has an open-source community behind and their libraries (DCMTK) make it possible for everyone to use it, modify and extend them. During this project DCMTK has been used to modify the DICOM data, for example, in the case there was data compressed with JPEG images, DCMTK has been useful to decompress it.

There are two main elements in the DICOM standard: on one hand the objects of images and information and on the other hand the operations that we can make with those objects.

The specification has a group of services and the definition of the object contained is in one class named Service-Object Pair (SOP). There, the informational objects are defined on the Information Objects Definition (IODs) and the features of one image can be defined: one can give information related with those objects or groups.

So the DICOM files are basically in files that apart from the file itself have information made in with the header of the file. The objects also can be compressed like we mentioned previously. When we have the DICOM, objects are stored in Picture Archiving and communication systems (PACS), that workstations are able to consult, download and store thanks to the DICOM standard. When we store the objects in these workstations are two kinds of devices: long-term and short-term storages, that means that long-term storages are permanently stored and short-term are only stored for a limited time (Yagüe, 2013) (Bidgood et al, 1997).

The DICOM that we received as data, was transformed to NRRD format because it would be easier to use it. NRRD is a library and a file format designed to support scientific visualisation and image processing in N-dimensional raster data. NRRD stands for nearly raw raster data, it is also flexible respect to type of integer and floating, encoding of written files (if it is compressed or not) and the byte order of data (Teem, 2014).
Example of a scalar volume NRRD file:

```
NRRD0001
content: aneurism
# Courtesy of Philips Research, Hamburg, Germany
dimension: 3
type: uchar
sizes: 256 256 256
spacings: 1 1 1
data file: ./aneurism.raw.gz
encoding: gzip
```

Table 2.1. Example of NRRD file

We can observe that the format is quite verbose and without seeing anything like that before we are able to understand it.

Among all the vtk that are available to use in the project, we have also used a model of the myocardium (heart muscle) of the same patient of the MRI. That mesh was created from the MRI, making a manual segmentation by the clinicians and saying what part of it is a heart and then applying an algorithm to create the mesh (for example marching cubes).

There are also some vtk files that are intra-cavity electrical mapping system data (CARTO), that means that it does not just have the topology of the mesh but also we have different look-up tables in which we could know the electrical activity of the points of the heart.

To get into what is the VTK format we have to know what the Visualization Toolkit (VTK) is, it gives us a lot of writer objects to read and write different and popular data file formats. It also provides some own file formats. There are two different kinds of file formats in VTK. The simplest are the legacy ones, serial formats that are easy to read and write. These formats are less flexible than the XML based file formats that support random access, parallel input and output, and portable data compression (Kitware, 2012).
Also we have data that there are electrocardiography (ECG) signals, but we have not used these kinds of data. These are the recordings of the electrical activity of the heart. The signal is recorded thanks to millions of individual cells and their activation (Noble et al, 1990). The files that contain this information are in txt and display the numerical values of the electrical activity and which is the activation of every point in each moment.

2.5. Applications of the visualisation in medical imaging

When we are talking of applications of the visualisation in medical imaging we have to mark that there are two main groups. These two groups are very different one from the other for one important feature, the connectivity, the need, or lack of need, of the Internet to use them. One group is about offline applications, it means stand-alone, and that we do not need Internet at the moment to use them. The other group of applications is the online, when we have the need of connectivity to Internet (or the server that provides the application) for its working.

2.5.1. Offline

a) medInria

MedInria is an application for the distribution of research software in medical imaging created within the Asclepios research project (https://team.inria.fr/asclepios/). The main goal of medInria is to have an application that is easy to use with a friendly user interface and with the last algorithms of the state-of-the-art with different ways of visualising medical images (Inria, 2012). It has six main features; database management and file import, visualisation, diffusion images processing, segmentation of images, registration of images and filtering of images. One visualisation of medInria is showed in the figure 2.4.
Another software developed by Asclepios is MUSIC (Multi-modality Platform for Specific Imaging in Cardiology). It consists in a platform for specific imaging cardiology that provides functionalities dedicated to it like the interventional planning and guidance. Some snapshots from the MUSIC platform are shown in figures 2.5.- 2.6:
b) MITK

Another application is MITK, which was used in the project for changing between formats of data. The Medical Imaging Interaction Toolkit (MITK) is a free open-source software system for developing interactive medical image processing software. The main features of this program are: interactive segmentation, the matching of two images based on two corresponding sets of points, the measurement of distances and angles, and the volume visualization (GPU-based) (MITK, 2014).
c) 3D slicer

3D Slicer is an application that is frequently used in medical imaging to analyse and interact with the data since prototype solutions is fast. It is an extensible platform being able to add plugs-in and with them, algorithms and applications. Some features are: robust dicom capabilities, interactive segmentation, volume rendering and extension manager among other things.

Figure 2.8. View of the 3D Slicer application [9]

d) Paraview

Paraview is an open-source, multi-platform data analysis and visualisation application. Paraview was developed for being able to analyze large datasets, but it can be run in supercomputers and as well on laptops (Paraview, 2015).
The paraview team is doing a web version of it that we will talk more about it in a next section.

### 2.5.2. Online

In contrast to offline application, the advantages of online are what connectivity can bring us, such as remote visualization of data, not having to to think about platforms and implement different kinds of the software depending of that or the sharing of content among others

a) **Slice:Drop**

Slice:Drop is a web-based application for visualising data in 2D or 3D. It uses WebGL and HTML5 Canvas to render in a browser. If the mobile device supports WebGL it will be able to run the application. It is based on the open source XTK toolkit. Slice:Drop shows the potential of the XTK library. It has a filesystem, image file dropping and medical image file formats are supported natively (Haehn et al, 2014).
b) Papaya

Papaya is the web version of Mango, which is a viewer for medical research images. Papaya provides analysis tools and navigation through image volumes. The main features are drag-and-drop, the coordinate type selection (between world space or index space), a feature that for a single image shows the info about it, adjust the minimum and maximum range, transparency and colour tables (University of Texas Health Science Center, 2015).
c) ScanView

ScanView is a browser-based tool for viewing medical images. It has annotation mode and different look-up tables implemented. This application is based on a modified version of the library XTK. It also uses BackBone, RequireJS, Twitter-Bootstrap, JQuery, JQueryUI, Underscore and ColPick (Basalla, 2015).

![ScanView Application](image)

*Figure 2.12. Example of the ScanView application [13]*

d) ParaViewWeb

ParaView has a version web of the application, ParaViewWeb, a web framework for 3D visualization and data processing. ParaViewWeb process large data and it uses ParaView for it, ParaViewWeb is the client and the images that are visualized are generated on the server-side and delivered them quickly. The majority of the applications are dealing with the visualization on the client side, and that uses to work for smaller data sets but not for large ones (Jourdain et al, 2010).
2.6. Enabling Technologies

a) VTK library
The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization and information visualization.

It includes support for 3D interaction widgets, two and three-dimensional annotation and parallel computing. VTK is coded in C++, so the users need to deal with objects to use the application.

VTK is the base of many applications such as Paraview, Visit, VisTrails, Slicer, MayaVi and OsiriX (VTK, 2015).

b) JavaScript
JavaScript was introduced in 1995 for adding an imperative programming paradigm to the otherwise declarative and static HTML web pages. It was first introduced in the Netscape Navigator 2.0 browser. After this, a standard, called ECMAScript, was written in order to control the evolution of the language. While there are still some differences, Javascript and ECMAScript are, to all intents and purposes, the same language.

Javascript is traditionally an interpreted language, which means that each line of code is parsed and executed at run-time, and we do not need to compile it. It is incorporated in all modern web browsers (including those on mobile and tablet devices), so it is a part of the web standard. The dynamic nature of JavaScript is that you can make applications that you can interact with, without having to reload every time the web page.

JavaScript can be used either on the server side or the client side, although the latter is much more used (Haverbeke, 2014).

c) 3D graphics on the web
3D graphics on the web is the use of 3D geometric data to make calculations and to create 2D images to be able to represent it to the screen. Rendering is the process to display data on the screen and all the steps that it takes.

The Graphic Processing Unit (GPU) is designed to process data in a parallel way permitting the execution of multiple operations (i.e. on more than one data point) at the same time. The power of the GPU consist in allow to access to the low level API such as OpenGL or Direct3D. The GPU is able to execute algorithms that are more complex in the same time than the Central Processing Unit (CPU), the reason is that they use parallel processing and not serial processing like the CPU. While the GPU is a processing unit whose primary use was for basic 3D rendering. Now they are more used to accelerate computational workloads while the CPU is a unit of processing dedicated to the non-graphical calculations (NVIDIA, 2015).

The fixed pipeline program uses algorithms that are predefined by the 3D programming API to calculate the position and colour in a scene. On the other hand, a programmable pipeline allows the programmer to define all of these things manually with shaders, programs that are compiled at runtime on the GPU (Evans et al, 2014). They exist to allow programmers to have direct control over the GPU at certain stages of the rendering pipeline. Shaders also have their own limitations in that there are some limits in how much information you can access in each step of the shader.

Depending on which part of the pipeline is overridden, there are different kinds of shaders, the most common ones are vertex shader (it is executed once for every vertex and permits to apply the transformations that convert it to screen space and do more operations before sending it to the pixel shader) and pixel shader (it is executed for every pixel that is painted on the screen and permits to define the color based in different calculations).

When we are talking about the differences of the architecture between these two components (CPU and GPU), we can affirm that the CPU is compounded of a few cores with cache memory that can handle a few threads at a time. On the other hand, GPU is compounded of hundreds of cores that can handle thousands of threads simultaneously. Figure 2.14 shows an image of the architecture comparison.
There are three main ways of visualising data on the web, one is browser-based rendering, and another one is remote rendering and different levels of combination of both.

In browser-based rendering it is the browser who performs the rendering, which means that it is the machine of the client.

Currently there are two main ways to paint in the client side. One is a declarative way, which means that you tell the computer what you want to happen and let it work out how to do it. Another way is an imperative way which means that you tell exactly all the steps that the computers has to take to achieve your result (Roberts, 2013).

Figure 2.15 shows a summary of which technologies are able to paint 2D or 3D and if they do so in a declarative or imperative way.
We are going to explain what is WebGL since it is the one that we will use. Like we see in the figure 2.15, it permits us to paint objects in 3D (also 2D) and it does it in an imperative way. It is a JavaScript library, which introduces an OpenGL derived 3D interface compatible with browsers without the need of plug-ins from any web page.

In the following paragraphs, the graphics pipeline will be explained from the bottom until the top, in order to understand all the steps that are required to paint pixels on the screen (see Figure 2.16).

At the lowest level is the Graphics Processor Unit (GPU) this is hardware and it does not have necessarily implement any specific API. The majority of the modern 3D hardwares are able to program individual units and modify the usual behaviour of the GPU, this is called shaders.

Over the hardware there is a driver that its function is to handle the low-level hardware aspects and a way to interact with it.

Above the driver is the scheduling, and its work is to share access to the GPU between different programs working.

And finally, on the top of all of it, there is the interface library, which is the access of the user to control the graphics of the machine (Context Information Security, 2011).

Observing the pipeline we can see that from the most abstraction level, the library (in the case of this application WebGL), we have direct access of the GPU and we can control what is going to be painted on the screen.

Figure 2.14. Summary of web technologies to paint [15]
When we talk about remote rendering means that the data is generated in the server side and then transmitted to the client to display it.

It is possible to classify the different approaches as:

- Graphics Commands: that means that low level draw calls are sent to the client which renders them and displays them,
- Pixels: that means that the server renders the image and sends it to the client for display it (like a video streaming)
- Primitives or Vectors: the most basic elements are extracted and sent to the client to visualize them, and there also exist combined techniques of this three (Evans et al, 2014).

Another thing to take into account is the data compression and the transfer. Medical data is usually relatively large in size, especially if the data is not optimized. So a lot of research effort is about mesh optimizations techniques.

However the main things to think about in this topic are (Evans et al, 2014):

- Rate distortion versus decompression speed, a technique that obtain high rates of compression may not the best solution for a web, since the decompression step is potentially slow, so with this problem you have a compromise and you have to choose what you need more.
● Browser-based decompression, in which way we decompress the data, we may be able to do it by JavaScript.
● Multiple scene objects, algorithms of progressive meshes may not work well with surfaces that are not closed or regularly sampled, so it can give some problems with some objects.
● Various data source of a single object, a mesh object might be storing more data besides the vertex which could be important for the displaying, such as normal, scalars that defines the colour, colour, texture coordinates, etc.

d) XTK library
XTK is a WebGL framework, easy-to-use API to visualize scientific data on the web. A summary of the XTK library is that has a native reading of a variety of scientific file formats, it has a volume rendering, thresholding and cross-sectional slicing of 3d image data. It supports label maps, color table, surface overlays and Constructive Solid Geometry (Haehn, 2015).

The way that this library is built and tested is following the system of CDash+ (web-based software testing server) and Google Closure that give us functionalities such as compile the code and then have a compressed version of it, increase the speed of the loading of the application, more easily cross-browser compatibility, etc.

The surface models/meshes files that are supported are: .STL (Standard Tessellation), .VTK (Visualization Toolkit polydata ascii only), .FSM, .INFLATED, .SMOOTHWM, .SPHERE, .PIAL, .ORIG (Freesurfer meshes) and .OBJ (wavefront .obj format).

The DICOM/volume files that are supported are: .NRRD (single file DICOM compressed and uncompressed), .NII (single file DICOM in NIfTI format uncompressed), .NI.GZ (single file DICOM in NIfTI format compressed), .MGH (single file DICOM uncompressed), .MGZ (single file DICOM compressed), .DICOM and .DCM (multi file DICOM uncompressed raw). The tractography files that are supported is .TRK. The scalar overlay files that are supported are: .CRV (freesurfer curvature measurements) and .LABEL (freesurfer labels). It also supports color tables and look-up tables files. The 2D texture files that supports are .PNG, .JPG and .JPEG (Haehn, 2015).
The interaction that the X Toolkit supports is the one of the mouse and keyboard and here is a summary of it:

<table>
<thead>
<tr>
<th>Mouse</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT CLICK + MOVE: Rotate the scene or Window/Level adjustment in 2D</td>
<td>ARROW KEYS: Rotate the scene</td>
</tr>
<tr>
<td>SHIFT + LEFT CLICK or MIDDLE CLICK: Pan the scene</td>
<td>SHIFT + ARROW KEYS: Pan the scene</td>
</tr>
<tr>
<td>MOUSE WHEEL UP: Zoom In, fast</td>
<td>ALT + UP: Zoom In, fast</td>
</tr>
<tr>
<td>MOUSE WHEEL DOWN: Zoom Out, fast</td>
<td>ALT + DOWN: Zoom Out, fast</td>
</tr>
<tr>
<td>RIGHT CLICK + MOVE UP: Zoom In, fine</td>
<td>ALT + LEFT: Zoom In, fine</td>
</tr>
<tr>
<td>RIGHT CLICK + MOVE DOWN: Zoom Out, fine</td>
<td>ALT + RIGHT: Zoom Out, fine</td>
</tr>
<tr>
<td>r: reset the view to the default based view or a manual configured camera position</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. Summary of XTK interaction

We have to mention that XTK uses the library Google Closure, this one provides some benefits like compile the code and then have a compressed version of it, increase the speed of the loading of the application, more easily cross-browser compatibility among others. This is important because it affects in all the developing of the code and follows some rules, but at the end you can decide if you compile the code or not.

e) Dat.gui
DatGui is a lightweight graphical user interface for changing variables in JavaScript in an easy way. We comment this library as we have been used in the application for the graphical interface.
3. **Methodology**

All the steps that take place in the design, preparatory work and implementation of the project are the mentioned right below.

In the design:
- Use cases
- Requirements of the application
- Study of how to address the design
- Decision about the application

In the preparatory work:
- Getting started with JavaScript
- Visualising and modifying data
- Visualising VTK data in the library XTK
- Visualising the MRI in the application

In the implementation:
- Summary of modifications on the XTK library
- Changing shader parsing method in XTK
- Creating the binary mask
- Adding a second render
- Process of finding the maximum and minimum value of pixel inside the mask
- Painting over the MRI using a mask
- Both renders are rotating and interacting at the same time
- Annotation on the application
- Connection with medInria
- Configuration file

Figure 3.1 shows the steps that the application takes to initialize all, after that we can interact with it.
It is important to point out which functionalities of the application are inherent to XTK, and which are additional elements, which represent novel, work of the project. Chiefly among the latter are:

i) a visual representation of the scar, interacting at the same time with two renders,

ii) going through the MRI and different slices with a mask associated,

iii) being able to load custom shaders,

iv) annotation on the application and

v) connection with medInria

In figure 3.2 we can see a diagram of the way that we interact with the application, and the functionalities of it.
3.1. Design

3.1.1. Use cases

The use cases are those that would allow a user of the medical application to:

- Share a segmentation result (not during the intervention) with an expert to have corrections via annotations. We will need the following functionalities:
  - Be able to visualise scar segmentation via thresholds and then allow the expert to change the thresholds.
  - Find the maximum intensity value inside the region of interest, in which the values above 60% will be called *scar*.
  - Be able to visualise at the same time grey-level images and segmentations
The data that will be visualised is a magnetic resonance image (MRI) and a mask that shows the region of interest. So the tasks that have to be done are visualising MRI 2D images in a 3D environment and a 3D model.

### 3.1.2. Requirements of the application

The requirements of the application have been agreed between the supervisors from UPF and medInria.

The functional requirements are that the application includes different ways to interact with:
- Interact with the data in different views, rotate and move them
- Change the window levels to apply different contrast
- Change the threshold of the scar
- Make annotations
- Export and import annotations

The non-functional requirements are:
- User can work with the data in medInria
- MedInria is able to export the data with its mask, and the maximum and minimum
- The data is uploaded to an FTP address, which works as a web server
- The developed application is to access to this data and give a link to access and interact with it

### 3.1.3. Study of how to address the design

As described above, XTK is a library oriented to medical imaging in WebGL. The first point to think about it is that if we want to make a web application in 3D. It will be necessary to have a canvas and modify it by JavaScript. The last advance to use is WebGL but it does not give us all the features to visualize the data, it is still necessary to code all the render and the visualization of the object.
XTK allows the users not to worry about to code and gives a context oriented to the visualisation of medical image since it has many parsers to different kind of data standardized in the medical environment.

There are more solutions to take into account, for example LiteGL or ThreeJS, but these two solutions are in a level below of programming of XTK since what they do is to make easier the management of WebGL. On the other hand they do not still have all the parsers or environment oriented to the medical data as XTK has.

Because of the lack of knowledge in these fields at the beginning of the project, we decided to make a first approximation to the medical environment and have a prototype, because of that the best decision for us was to use XTK.

### 3.1.4. Decision about the application

The main purpose for this project it is to help collaboratively and exploit the features of the web.

Some functionalities decided from the meetings are: changing shader parsing method, adding a second render to be able to interact with two kinds of data at the same time, painting over the MRI using a mask, annotation in the application and connection with medlnria.

### 3.2. Preparatory work

#### 3.2.1. Getting started with Javascript

The first thing that is important to understand is basics of the client server web-architecture, and understanding where Javascript processing takes place. To get used to all of that, during the beginning a simple chat application was made.

As it is explained in section 2.7, Javascript is a language that is more commonly used in a browser, so it is the one that interacts at the client side of the communication, this side can ask for files or operations to the server side. The server can be programmed in any other language that does not have to necessarily be JavaScript. Nevertheless, servers are usually programmed in JavaScript language, but other possibilities are also factible.
In order to speed up the learning process, XAMPP software was used to host the server locally.

![Diagram of client-server communication](image)

Figure 3.3. Example of the communication between the client and the server [19]

The communication process is the following: the client sends an HTTP request with the url that he wants to access to. When the server receives the request of the client, it sends a response with an HTML document. You can find a clear example of how this works in Figure 3.3.

![Diagram of client-server communication](image)

Figure 3.4. Examples of different application model [20]
3.2.2. Visualizing data

We are only going to show and explain the data that is used in the application since it is the more important topic and the one that gives us more information.

We can observe a MRI of a patient with atrial fibrillation disease in the figure 3.5 and a cardiac model of the heart of that patient in the figure 3.6. This 3D model was extracted by manual segmentation of the MRI.

![Figure 3.5. Visualisation of the MRI of a patient with an atrial fibrillation disease in MITK [21]](image)
The cardiac model showed in the figure 3.6 contained both the left and the right ventricle. Since our region of interest was only the left part, we removed the right one using Paraview.

To remove the right atrium we should follow this process (Paraview, 2008):

1) First of all, download a previous version of Paraview 4.0, since the function we need was removed in the fourth version and it is not available anymore.

2) Import the data that we want to Paraview

3) Open the Selection Inspector and select the points that we want to remove of the mesh
Figure 3.7. Selection of some cells of the right ventricle [23]

4) Check the box -> Invert selection

Figure 3.8. Zoom on the selection inspector [24]

5) Apply the Extract Selection filter
6) Being that XTK does not compute normals as Paraview does, we will create them applying the filter *Generate Surface Normals* to export them with the mesh.

7) Export the new mesh.

3.2.3. Visualising VTK data in the library XTK

First of all, we need a html file where all the libraries, included XTK, are loaded. We also need a css file that tells us how the webpage should look like. For example, it
would show how a black background would look on it. Finally, we need a JavaScript file where we code all the steps to tell the browser how to paint the data. The steps to do that were the following:

- Create and initialize a 3D renderer
- Create a new mesh
- Set the path where the mesh is
- Add the mesh to the renderer
- Finally, we render all the objects

The data that we have used is a CARTO mesh, in which one has information of electrical points in the mesh, made of different look-up tables. We implemented a way to visualise the look-up tables and the colors associated to them. With this implementation we are able to change from one to another using the menu. We can observe them in the figures 3.11 - 3.13. This implementation was not used in the final application but we think that it will be useful to future functionalities.

![Figure 3.11. CARTO data without any lookup table](image-url)
3.2.4. Visualising the MRI in the application

Before the visualisation of the MRI in the application, we had some problems with the data that we had to fix to finally be able to display it. The MRI was given in a directory with 112 dcm files (a DICOM format). The problem to read this data directly in XTK is that the data is compressed in JPEG image and XTK does not support it at the moment.
We fixed this problem by converting the format to one that XTK is able to support. To do that what we had to open the data in the MITK application (figure 3.14) and save it in NRRD format.

Finally, when we had the data in that format, XTK was able to support it. We needed an html file where all the libraries, included XTK, were loaded. In addition to that, the html had to create different containers (sections) for the different renders that the application was going to have at the end. These sections were for the 3D environment, slice X, slice Y and slice Z. Also a css and finally a JavaScript file where we coded all the steps to tell the browser how to paint the data. The steps for doing that were the following:

- Create the 3D renderer and initialize it
- Create the 2D renderers for the X orientation, for Y and for Z
- Create a volume
- Attach the single-file DICOM in a .NRRD format (the data that we converted before)
- Add the volume in the main renderer, the one that deals with the X slices (we chose this one in case that WebGL is not supported because it should work)
- We start the loading and the rendering process of the main renderer
- In the main renderer we use a method called onShowTime. It gets executed after all files are fully loaded and just before the first rendering attempt, and what is done is:
● Add the volume to the other 3 renderers, these are slice Y, slice Z and 3D
● Create an object to manage the GUI that is done with Dat.Gui
● Configure controllers which switch between slicing and volume rendering
● Configure the volume rendering opacity
● Configure the threshold in the minimum and maximum range
● Configure what index X, Y or Z are the currently displayed slices in the range from 0 until dimensions - 1

Figure 3.15. MRI on the XTK application [31]

3.3. Implementation

3.3.1. Summary of modifications on the XTK library

We had made some modifications on the XTK library, since we noticed that we need more functionalities that the ones that XTK provided.

Figure 3.16 shows a simplified diagram of classes in XTK.
The classes that we had to modify were:

- parserNRRD: make that the decimals match between different calculations
- parserVTK: read more data and save them for a lookup tables
- mesh: implement a function for changing colors and another for interpolate colors for show lookup tables
- object: added a boolean variable for know if that object is a mask
- slice: implement getters to get additional information that before was not accessible
- volume: add a variable to indicate that something is a mask, implement getters for access to more information
- camera3D: implement a getter of the perspective matrix
- renderer: implement a getter for the camera and the context of WebGL
- renderer3D: add variable for default shader, added code for being able to load custom shaders, added code for being able to paint the mask by the shader, this is explained in section 3.3.6
- shaders: add variables in the shaders to make the code work properly with the modification of the custom shader
3.3.2. Changing shader parsing method in XTK

One functionality that we think that is useful is to load custom shaders or different ones than the ones that the library XTK has. Since in the current library XTK is not available for now, that means that you can only use the one that comes by default with the library and these ones are hard-coded (typed in the code without external files).

Now we will see in which way the loading of the custom shader is achieved in XTK. First of all we create a new class that is called ShaderLoader. In that class we have the constructor of the class and two functions loadShaders and loadFromFile.

- *LoadShaders* receive 3 arguments, urlVertex, urlFragment and callback, urlVertex is expected to be the location of the file that is the shader vertex, urlFragment is expected to be the location of the file that is the fragment shader and the callback is a function that will be executed when we will want. In this function what we do is to call twice the function that we will explain below (loadFromFile): one with the urlVertex argument and one with the urlFragment argument.

- In the function *LoadFromFile* what we do is to request the location of the file, read it and save it in shaderObject that we have created at the beginning. We do this for the vertex shader and for the fragment shader, once these two calls are complete, we execute the function that we had for argument (the callback).

For the fully working of the application, the callback function has to be the initialisation of the rendering, which inside of it we will replace the hard-coded shaders for the ones that we saved from the specified locations.

3.3.3. Creating the binary mask

No mask matching with the resonance was provided. However, it was a model in 3D, so we had to create the binary mask in the VTK library in an offline way. Since the VTK library is in C++ we used that language for make it.
Before going into the programming of the new volume, we have to know all the features of the volume that we want to match, to know all of these, we used MITK and look at the details of the DICOM file.

![Screenshot of the data displayed of the MRI in MITK](image)

**Figure 3.17. Screenshot of the data displayed of the MRI in MITK [33]**

The data that will be useful for us, there are the Dimensions, the Origin and the Spacing.
First of all we had to read our vtk model of the left ventricle (the one that we removed the right ventricle in the section 3.2). Next to that, we create a filter and depending on the version of VTK we send the data of the vtk file in a different way and we update it. We create a volume where to put all the slices, and we put the desired volume spacing, this was a significant value to have into account if we want that the slices perfectly match between them, we looked at this information in the figure 3.17. Also, the desired dimensions and the desired origin both of them are in the figure 3.17.

We fill the image with foreground voxels, with foreground we mean the ones that are interesting and inside the 3D model, and voxel is a pixel in 3D. We create an object of type vtkPolyDataToImageStencil that does exactly what we need, converts a 3D model into a volume representation, where we will paint the foreground pixels in 255 (white) and the background pixels in 0 (black). In the figure 3.18 we can see an example of what this class does with a sphere.

Next to that we input the vtk data as the input of the filter (we make it different depending on the VTK version).

![Image](image.png)

Figure 3.18. Example of what does the vtkPolyDataToImageStencil filter [34]
We put the origin, the spacing and we extend the filter to the right dimensions. We cut the white image and set the background depending of the vtk version. Finally we save the volume to disk in Metaimage file format (mhd).

We had some issues while we were making the binary mask. One of them was that the origin of the mask. The slices did not completely match between them, the bounding box did not match at the beginning and also we did not know the exact spacing. But all these problems were fixed looking at the data and putting them manually.

We can see a link to the code that we looked in the references (Paraview, 2012).

**3.3.4. Adding a second render**

A second render is added to the the application, in which the 3D model of the left ventricle is displayed. The decision of creating a second render to visualise the model and do not use the initial one was taken into account because we thought that in some cases, visualising data superposed could be overwhelming as it was too much information together: that disturbs more than it helps as some people related to medical environment advised us. So separating the data for a better visualisation and avoiding overlaying them was the best approach in this case. In the figure 3.19 we can observe how all the renderers look like.

Since we have been displaying this kind of data before (explained in section 3.5) what we have to do is create a new section in the html document to place the new render. Just like we said in some occasions before, this data source was created from the MRI by doing a manual segmentation of it. At the beginning it has the left and right ventricle, but we decided to split it. What we did next was create the render and associate it to that container, initialise it, add the elements that we wanted to display on it (in this case only the model one) and render it. To correctly place the object in a centratred position within the render, we also put it in the same coordinates as the first render, we flipped the X and the Y and we translated the object to the right point of focus.
Figure 3.19. Screenshot of all the renders added into the application [35]

3.3.5. Process of finding the maximum and minimum

In this section we will see the process of finding the maximum and minimum value at the region of interest (inside the mask).

The mask is a volume that has the same dimensions that the volume of the MRI. Also every slice of one volume corresponds with the other. The mask consists in two values 0 (black) and 255 (white), the value 0 (black) means that the pixel is not interesting and we should not take in account what value has in the MRI, instead the value 255 (white) means that the pixel is on the region of interest and we should take into account that value for find what one is the maximum and minimum.

The steps that the algorithm takes are:
- For each dimension (there are three, sagittal, coronal and axial)
  - For each slice
    - For each pixel
- Evaluate if the pixel is in the region of interest, if it is check if it is the highest or the lowest value, if it is that case, save that value
- Return the maximum and minimum value

3.3.6. Paint over the magnetic resonance using a mask

These are the steps of the algorithm to paint over the MRI using a mask:
- Load custom shaders
- Load the MRI
- Load the mask
- Associate the MRI with its corresponding mask
- Send the actual slices of the mask and the MRI that are rendering in that moment to the GPU (these slices are working as textures)

From this point to advance we will see the programs that are being executed in the GPU. In the figure 3.21 is showed the pipeline of the GPU and some shaders that can be used, in our case are the vertex and fragment shader.

The first “program” that is executed is the vertex shader, in this program we have information of the rendering process that is the vertex position, vertex normal, vertex color, vertex texture position and a scalar associated to the vertex.

There is a special kind of variable, called varying, which are used to send information to the next program that is going to be executed on the GPU, fragment shader.

The steps made on the vertex shader are:
- Define the point position
- Define the point size
- Send the varying data to the fragment shader

The next program to execute is the fragment shader. We receive the following data from the previous program: if we should discard the pixel, vertex position, color pixel, texture position, vertex normal and the transformed vertex normal.

\[
value > (\text{maximum value}) \times (\text{threshold percent unhealthy})
\]

Figure 3.20. Formula for consider a region unhealthy [36]
The steps made on the fragment shader are:

- Evaluate if we are in discard mode, if we are, we finish to execute the shader
- Evaluate if we are in picking mode, if it is yes, we paint the fragment in the color indicated
- Evaluate if we are using a texture and in case that is a volume one we do:
  - Check if we have a mask, if is yes:
    - Check if the fragment is in the region of interest
      - Evaluate if the fragment is in the values that indicates that is an unhealthy region (go to figure 3.20 for the formula)
      - Evaluate if the fragment is a contour, if it, paint it white
    - If is not a texture or any other case, apply a lighting to the object

An example of the final result of painting the unhealthy region and the contours is in the figure 3.22.
3.3.7. Both renders are rotating and interacting at the same time

We think that it is important to make that the two renders of the application; the one that is displaying the magnetic resonance and the one that is displaying the mesh of the left ventricle, are interacting in the same way with the user. Since it is the same patient, but we are displaying different kinds of data. The reason of why we made this is that if the interaction is in the same way (rotations and movements) it will be better to understand the information and not get lost into the data, because it can be a bit confusing having to move the data two times for them to be synchronized again. Screenshot of the application in the figure 3.23.
To be able to do this we make that the render that is displaying the mesh has the same center and camera of the render that is displaying the magnetic resonance. If we stop at this point what will happen will be that when we interact with the render of the magnetic resonance the other one will be moving as well, but if we move the second one, the one of the magnetic resonance will not be doing anything. To fix this issue what we have to do is that the render of the magnetic resonance has to take into account the interaction of the other render.

To do that, we have to make that the render of the magnetic resonance observes the interactor of the other one, then when we will move any of one render the other one will be moving also.

### 3.3.8. Annotation on the application

a) Workflow for annotation

The annotation module works in the following way:
- We indicate that we are in the annotation mode, for disable all the rotations and movements
- We click on the data and drag the mouse where we want that the annotation line ends
- The application will request for some text for the annotation
- The annotation will appear in the 3D environment and will rotate in the same way that the data does
- The annotation (or annotations) in the actual 3D environment can be saved in an external file and reintroduce them later

b) Setup

There was a problem with the XTK library in terms of implementing the annotations, and that is that we can not calculate the intersections with the XTK library because it does not give the exactly position where the intersection was made, only if the object was intersected or not.

That one is the main reason to use another library that helps to achieve picking via raycast. So the approximation used to implement, it was to use an existing library that
Javi Agenjo, member of the GTI made. That library is litegl.js, is a library that wraps WebGL in a way that is more user-friendly to use it (Agenjo, 2015). We connected the existed canvas that the XTK library was using with the one that the new library was going to draw in it.

c) Ray cast picking implementation

The next step is to create three planes according to the planes that are shown in the volume (see figure 3.24), that is because the best approach is to recreate the scene geometry in the second library to mimic the XTK geometry and then run the algorithm to calculate the intersections.

![Figure 3.24. Visualisation of the three planes](image)

When we click the mouse, the first thing that we have to do is to check if we are in the annotation mode and if when we clicked the mouse we had an intersection with an object. That is made using a ray that is cast from the camera position to the position of the mouse pointer on the screen. The intersection point of this ray with the planes (slices) in the scene is calculated and stored as the annotation start point.

When we release the mouse we make that another ray is cast from the camera position to the new position of the mouse pointer on the screen and we calculate the intersection of the ray with a new plane that is made of the point of the first intersection.
When we have the two points of intersection (clicked and released mouse), we create a line between that two vertices and we draw it into the environment. An example of one line of annotation in the application is on figure 3.25.

![Figure 3.25. Example of the line of annotation in the application [41]](image)

d) Text in the annotation
The next step is to make that the user can put text to that annotation. So when we release the mouse it will appear a window to introduce the text (figure 3.26). This was achieved using a function called prompt in JavaScript.

![Figure 3.26. Window to enter the text into the annotation [42]](image)
e) The annotation moves in the 3D environment
The main difficulty for do it was that we had to calculate the new positions in the screen every time that we have moved in the environment, this was achieved calculating continuously the position on the screen with the matrix that projects the point from 3D to the screen, and then update the annotation, an image of how it looks is in the figure 3.27.

![Figure 3.27. Annotation in the application [43]](image)

f) Saving annotation to the server (for collaborative work models)
There is a way to be able to export and import annotations, in that way we are going to be able to send them and import them later, with this we address the remote collaboration, one of the objectives of the project.

The one for exporting takes a list of all the annotations and use all the necessary information for being able to redo the annotation. What we do in this function is to convert the list in a JSON file (open standard that uses readable notation to transmit data objects) and write it in a file.

The one for importing takes as parameters the JSON Text that contains all the information, the information needed to redo the annotations are:
We load all this data into the application and we create new applications with it.

### 3.3.9. Connection with medInria

When we thought about the possibilities of the connection with medInria, three main options appeared:

1) Conversion is done 100% client side. The user submits data locally in the browser, the conversion is done client side, and the server does not necessarily do anything. All code will have to be written in Javascript. The pros are that we do not have to deal with client-server links and bandwidth issues. And the cons are that it is needed to write conversion code in JavaScript; no server connection means that each client has to do conversion individually.

2) Conversion is done 100% server side. The user uploads the entire file to the server, where a process does the conversion, and posts the result back to the client. The pros are that the full converted file is stored on the server for all users; it can be used a C++ process to do conversion. The cons are that it uses loads of bandwidth

3) Metadata conversion done on server. A JS library parses the input file and extracts the metadata, and dumps the actual data into some buffer. The metadata is sent to the server, which then does some conversion. The JS library reassembles the XTK file. The pros are that has the best benefits of 1) and 2). The cons are that it is still needed to have to parse the input file in Javascript, so it is not much less work than option (1).

The option what we thought that was the best one for our application is the version one because the pros that can bring were a better approximation for us.

The ideal workflow of the connection between medInria and the application is:

1) User selects scan to work on

2) User segments region of interest of the dataset using Medinria

3) Medinria calculates max-min values within region of interest
4) MedInria exports 4 files:  
   i) NRRD file of dataset  
   ii) NRRD file of region of interest binary mask  
   iii) VTK mesh of reconstructed from binary mask  
   iv) An JSON file which references each of the above file names, and also stores the max-min values from step 3), this is a configuration file  
5) All this data is uploaded to a folder at an FTP address, which also doubles as a web-server of the application  
6) The Remote user can use the tool implemented in this project to visualise the dataset and comment, annotate and interact with the data

3.3.10. Configuration file

A configuration file to load all the data and parameters into the application has been created. This makes easier the loading of contents and the use of the application, since if we want to change the file to load or the annotations, we do not have to change the code, we only have to change the configuration file. The actual configuration file is shown right below.

```
[  
   {"urlMR": "data/rmiTFG.nrrd",  
    "urlMaskMR": "data/CardiacModel/CardiacModelLeftVentricleHugeNewOrigin.nrrd",  
    "meshVentricle": "data/LeftVentriclePolyData.vtk",  
    "maxMR": 34,  
    "minMR": 0,  
    "annotationToLoad": true,  
    "annotationUrl": "js/logAnnotations.json" }  
]
```

Table 3.1. Example of the configuration file
4. Results

4.1. Final application

The final application is a web based application in which we can interact with the available dataset (figure 4.1), interact in real time with MRI images and make annotations. The best browser to use it is Google Chrome. A prototype of the final application is available in the following link: http://webglstudio.org/melanie/index.html

Figure 4.1. Final application [44]
There are two kinds of data:

<table>
<thead>
<tr>
<th>MRI from one patient with an atrial fibrillation disease</th>
<th>3D model of a heart that was done by a manual segmentation of the MRI</th>
</tr>
</thead>
</table>

![Figure 4.2. MRI data [45]](image1)

| Figure 4.3. 3D model of the left ventricle [46] |

![Figure 4.3. 3D model of the left ventricle [46]](image2)

Table 4.1. Summary of data in the application

There are different modes to interact with the data. In the 3D environment, you can zoom in and zoom out using the wheel of the mouse or, in case you have a pad, in the equivalent way. You can click the environment and all the data will rotate as you are moving the mouse. The keyboard combination Shift + click and drag will allow us to displace the 3D data.

Figure 4.4 shows three orthogonal views (sagittal, coronal and axial) of a MRI dataset.

![Figure 4.4. Planes of the slices [47]](image3)

There is a mode in which you can press Shift + pass over the mouse and the other two slices will change (depending in the direction you move the mouse to: you can move it up and down or left and right) and let the one that you are interacting with fixed. Pressing Shift + click and drag will move the image.

In the right part of the page we can observe a menu section (see Figure 4.5) with some parameters to interact with:
- visible: if we want the volume to be visible or not
- sliderScar: modifies the threshold of what we consider to be the scar. 0.6 (60%) is the standard one. If the value is above the 60% of the maximum the application will consider that region as a scar, and it will be painted red.
- windowLow: it decreases the contrast when it is high, it makes the minimum value to paint the selected one.
- windowHigh: it increases the contrast when it is low, it makes the maximum value to paint the selected one.
- Sagittal plane: it is a slider to select which is the sagittal we are in. We can change it to switch the plane to 3D environment and to the images at the bottom of the page.
- Axial plane: same as before, but axial.
- Coronal plane: same as sagittal and axial, but in the coronal case.
- Annotation Mode: if annotation mode is selected, the rotations on the MRI data will be disabled. Instead, we are able to make annotations. All we have to do is clicking on the data and drag the mouse until the point where the line of the annotation should end. Then a box will appear: there, we can insert the text and it will be showed as an annotation.
- Visible Annotation: if the checkbox is checked the annotation will be visible, if not it will not.
4.2. Feedback

Thanks to the medInria people it has been possible that one clinician gave feedback about the application. The main aspects that he emphasized were:

- The display of mesh contours of scar and myocardial boundaries in the 3D was nice and clear, although it would be better that the user has the ability to put two thresholds on scar in order to also visualize grey zone, which would imply to add some translucency to scar and grey zone areas.

- He was enthusiastic about the annotation functionality. An advice to make it even better was that might be nice to have the full list of annotations on the right side of the screen, and an automated positioning of the three planes intersection on each annotation site once you click on it.

- On the application the annotations are visible even if they are located behind a plane. This could be confusing if there are a lot of annotations. An advice is to hide the text when it goes behind.

- The scar segmentation was not accurate enough due to the inaccurate endocardium segmentation. That problem using the annotation functionality might get complicated to use the tool since is better to refer to regions rather than points. One solution proposed was to be able to define a region.

- He thought that the upper right panel would only make sense if the scar was visible within the myocardium mesh.

- He thought that the shift + pass over function in the three planes displayed at the bottom was really nice and he had not seen something like that before.

- One advice that he made was that if we want to make the use of the application easier for cardiologists, it would be good to put shortcuts to the common orientations used in cath labs.

Even if a single opinion is not enough to evaluate the application, knowing the opinion of professionals in this field gives an idea of whether the developed application can be useful.
5. Conclusions

5.1. Evaluation

We think that the final results are positive because we developed an application that is a prototype of something useful. We achieved the principal objectives of the project. These were to foster collaboration using a web-based visualisation interface, accessible on any platform which has access to a modern web-browser and from any location, allowing the sharing of content between remote users and permitting them to give feedback and to discuss 3D multimodal data.

Some of the functionalities planned to implement (such as making the scar visible on the 3D model), were finally not developed due to time constraints.

The contribution of this project is an online application that is used to connect a stand-alone application and share data in it. This is in contrast with the other two applications developed with XTK (Slice:drop and ScanView), since none of them has this functionality, they are only addressed to load data online and they do not have any connection with a stand-alone application.

We are going to summarize briefly the most implemented functionalities. First is that we are able to interact with the image in real time, that means that we are comparing values in the web on the region of interest. Second and related with the functionality mentioned before, we can, given the maximum in the region of interest, calculate what part of the image is scar. We could be making more interaction and calculation with the image since we are doing it by the GPU and is faster than the CPU, this could be done due to the new form of load custom shaders. One of the functionalities that we have and that is very useful is to make annotations in a 3D environment and be able to share them, this functionality has been more exploited by other applications, for example in ScanView. Another functionality is that we are able to export the annotations, even that, this functionality is not visible for the user but there is a way to do it by the console of the browser.
And finally one functionality that is very important but it is not visible for the user is that we have a method to load the paths of where the data is and what files we have to load into the application, this is very important because we can configure the scene, specify the maximum in the region of interest and change the files to load without having to change the code.

5.2. Future work

The future work of this application is linked to the future of XTK. The developers of XTK commented that they already noticed that the library was reaching some limitations with the flexibility and accuracy. They also commented that fixing all of that require a lot of work.

They were thinking about make a migration of the code to use ThreeJS (a JavaScript 3D library) to power all the visualization and then be able to to focus on the medical imaging part. The main objectives will be support all the file formats, VTK/STL models, single channel, multichannel, time series, etc, in a better way, fix the orientation issues and that all the calculations are correct. One new thing that they want to implement is new items as widgets to be able to implement new functionalities in a easy way. They also had been fixing internal issues to make xtk more efficient\(^1\).

Therefore, to continue with the project, the best approach will be first to migrate the new implementations and modifications to the new library with its own new approximation of the WebGL context. Then, keep implementing use cases and fix problems with the actual application.

We mention right below some use cases and functionalities which could be implemented to extend the project.

These are the use cases:

- Share a co-registration result (not during intervention) with an expert to have corrections via annotations.
  - Need for visualising two meshes at the same time with opacity/transparency options

\(^1\) [https://github.com/xtk/X/issues/177](https://github.com/xtk/X/issues/177)
The data would be a segmented MRI and data CARTO. The application should also show both of them at the same time.

- Share the current data available during the intervention with an expert to get suggestions for ablation targets
  - Need for visualising CARTO and pre-operative MRI data. It would also be useful to be able to visualise 1D signals for each CARTO point
  
  The data used in this user case would be the same that in the second user case.

There are some functionalities that could be implemented like having a real-time update of scenes on multiple viewing devices and having a connection by camera/microphone with different users.

Other future work is to collect feedback from different users, analyze that information and make design decisions from it (e.g., functionalities interesting to implement, errors that could appear in the application or better ways to address to the user).

### 5.3. Personal evaluation

My personal evaluation of this project is that it has been hard. This is because trying to do something of your own and deciding how to address it, while accepting that you can be wrong or right in your decisions, is something tough to deal with.

When you have never done a long-term project, it is a bit scary because you do not see the end of it, and you do not know if you will be able to achieve it.

Also it was hard because it has included different supervisors and people from a foreign company, which made difficult to combine all the schedules for everyone and be able to communicate and agree among them. This was partially due to the fact that there were people of different backgrounds.

In addition, writing the report in English has been hard because I do not consider that my level of English is high enough, and because it is a scientific report. However, I choose the English language to start to get use to it and because in that way it could reach more people. Thus, I do not regret the added difficulties.
Even all the difficulties and the fact that I know the developed application is not perfect and needs a lot of work to make it better, I have learnt a lot developing it and I think that was a great experience and necessary to end the degree.

One of the main motivations of doing a project that tries to help in the medical environment is personal. One of my parents died of a cardiac disease a few years ago and I would like to help to contribute. Thus, the fact that I could have some prototype of something useful at the end and that a clinician thought that was a useful and interesting tool made me glad that I chose this project and I would like to continue with it or in this kind of field.
Bibliography


(American Dental Education Association, 2012) American Dental Education Association (2012), http://explorehealthcareers.org/en/Field/2/Informatics (Seen on 8th of May)


(Paraview, 2008) Paraview, (2008), Data Selection http://www.paraview.org/Wiki/Data_Selection (Seen on 9th of May)


(Paraview, 2012) Paraview, (2012), Poly Data To Image Data http://www.paraview.org/Wiki/VTK/Examples/Cxx/PolyData/PolyDataToImageData (Seen on 9th of May)


(University of Texas Health Science Center, 2015) University of Texas Health Science Center, (2015), Papaya http://ric.uthscsa.edu/mango/papaya.html (Seen on 15th of May)

(The University of Queensland, 2014) The University of Queensland, (2014), Introduction to Biomedical Imaging https://courses.edx.org/courses/UQx/BIOIMG101x/1T2014/ (Seen on 3rd of April)


http://www.ncbi.nlm.nih.gov/books/NBK354/ (Seen on 10th of May)


Appendix

Figure links

[9] https://wiki.canterbury.ac.nz/display/BlueFern/ParaView
[16] http://www.paraview.org/Wiki/VTK/Examples/Cxx/PolyData/PolyDataToImageData
[19] [20] http://www.paraview.org/Wiki/VTK/Examples/Cxx/PolyData/PolyDataToImageData