Inflammation evaluation through low-cost 3D scanning of human body parts

Guillermo Girona San Miguel

Defense date: January 28th, 2015

Director: Carlos Andújar Gran
Computer Science department

Course: Degree in Informatics Engineering
Specialisation: Computing

Centre: Facultat d'Informàtica de Barcelona
University: Universitat Politècnica de Catalunya - BarcelonaTech

Abstract

Some conditions of the body such as rheumatoid arthritis are characterised by inflammation of one or more joints. The detection and evaluation of the degree of inflammation is usually done by professionals such as rheumatologists through clinical exploration and medical imaging (ultrasounds and MRI). The aim of this project has been to explore different techniques to evaluate the amount of swelling of a body part through optical 3D scanning.

We have created a computer program that is capable of loading 3D models acquired from real joints. Using different techniques like alignment of models, visual comparison and calculation of different measures like distances, areas or volumes our program provides valuable information to facilitate the assessment of the inflammation level, using data from a symmetrical joint or from the same joint acquired at a different moment.

In order to acquire the models that represent the shape of the skin surrounding the joints to be analysed we have used low-cost technologies such as light-based 3D scanning or depth cameras.

Resumen

Algunas condiciones del cuerpo como la artritis reumatoide se caracterizan por la inflamación de una o más articulaciones. La detección y la evaluación del grado de inflamación suele ser realizada por profesionales como los reumatólogos a través de exploración clínica e imagen médica (ultrasonidos y resonancias magnéticas). El objetivo de este proyecto ha consistido en explorar diferentes técnicas para evaluar el nivel de hinchazón de una parte del cuerpo a través de escaneado 3D óptico.

Hemos creado un programa informático que permite cargar modelos 3D adquiridos a partir de articulaciones reales. Usando diferentes técnicas como la alineación de modelos, la comparación visual y el cálculo de diferentes medidas como distancias, áreas o volúmenes nuestro programa proporciona información valiosa para facilitar la evaluación del nivel de inflamación, usando datos de la articulación simétrica o de la misma articulación en un momento distinto.

Para adquirir los modelos que representan la forma de la piel que rodea a las articulaciones que se quiere analizar se ha hecho uso de tecnologías de bajo coste como el escaneado 3D basado en luz o las cámaras de profundidad.

Resum

Algunes condicions del cos com l'artritis reumatoide es caracteritzen per la inflamació d'una o més articulacions. La detecció i l'avaluació del grau d'inflamació acostuma a ser realitzada per professionals com els reumatòlegs a través d'exploració clínica i d'imatge mèdica (ultrasons i ressonàncies magnètiques). L'objectiu d'aquest projecte ha consistit en explorar diferents tècniques per a avaluar el nivell d'inflor d'una part del cos a través d'escanejat 3D òptic.

Hem creat un programa informàtic que permet carregar models 3D adquirits a partir d'articulacions reals. Utilitzant diferents tècniques com l'alineació de models, la comparació visual i el càlcul de diferents mesures com distàncies, àrees o volums el nostre programa proporciona informació valuosa per a facilitar l'avaluació del nivell d'inflamació, utilitzant dades de l'articulació simètrica o de la mateixa articulació en un moment diferent.

Per a adquirir els models que representen la forma de la pell que envolta les articulacions que es vol analitzar s'ha fet ús de tecnologies de baix cost com l'escanejat 3D basat en llum o les càmeres de profunditat.

Contents

1 Introduction

The use of computer applications is becoming a very common practice in the modern society. Mobile devices like smartphones or tablets have become very popular, and that has led to the creation of applications that solve any kind of problem that one can imagine. As it could be expected, there are applications that help a person to know if he or she is in good health. There are applications that measure the heart rate, applications that help the user to keep a healthy diet and applications that give indications of how to act in a medical emergency.

In this project our aim has been to develop a tool that given data of the same body part in different states of inflammation compares them in order to give the user information about the evolution of the inflammation. Using our application a patient or a professional will be able to compare a joint with its symmetrical one, or with the same joint in a past state.

The data used will be a 3D model of the body part that suffers from inflammation, thus allowing us to make use of 3D graphics and virtual reality techniques in order to give the user the feedback that he or she needs.

This document aims to give the reader a detailed description of all the areas of the project. First of all, a description of the project planning can be found, giving details about the scope, the context, the temporal planning and the economic planning, in the appropriate sections. After that, sections devoted to the development of the project can be found, detailing the implementation of the different pieces of code that are part of the project and the techniques explored. Finally, there are two more sections, a report of the tests performed and the final conclusions.

2 Scope of the project

This section aims to make a description of the scope of our project. We are going to start with the formulation of the problem that we have solved. Then, the scope of the project is going to be defined. Finally, we are going to describe how we have worked to solve the problem in the *Methodology and rigour* section.

2.1 Objectives

The main objective of our project has been to develop a tool that given data of the same body part in different states of inflammation compares them in order to give the user information about the evolution of the inflammation. Our aim has been to design the tool in a way the user finds easy to track the evolution of his or her inflammation, providing him or her enough information in a way it is intuitive to read and understand.

A secondary objective has been to explore and implement algorithms capable of comparing the data representing the different states of the same body part in order to be able to give the user useful feedback.

The data used is going to be a 3D model of the body part that suffers from inflammation. In order to do that, we defined another secondary objective of the project, consisting in using different 3D scanning techniques to obtain the 3D model of a body part.

2.2 Scope

In order to be able to solve the problem that defines our project we needed a program capable of loading two or more 3D models of the same body part and comparing them, showing the user the differences caused by swelling, which is a direct effect of inflammation. So, we have designed a 3D viewer with more features than a common viewer would have, but since it is not the most important part of the project, we have not explored advanced 3D features. We have tried to create a program capable of showing the information the user needs in an intuitive way, but nothing else.

Algorithms to compare different 3D models that represent the same physical object, but

possibly having a slightly different shape were needed. Our project required at least a working algorithm, but we have explored different algorithms in order to compare them and even offer the useful ones to the user. These algorithms provide information about the variation of the inflammation level of the body part, and some of them provide information about the level of inflammation of each zone of the body part. Furthermore, we have found an algorithm to automatically match the different models that represent the same body part, refining the manual alignment done by the user.

Finally, we needed a way to obtain the 3D model of a given body part. In order to do that, we have explored different low-cost techniques of acquiring a 3D model of a physical object. At the start of the project we already had a working technique, which was the use of a light-based 3D scanner, considered as a low-cost technique because even if the price of the scanner is high, it can be used plenty of times, so the price per use is low. It has a problem, though, since not everyone can have access to this kind of scanner. That is why have explored techniques that are available for everyone. We have proposed using a *Kinect*, the motion sensing device originally designed for *Xbox*, which is a device that a great amount of the people that owns a *Xbox* console has, and which has got extensive support for software development. Another proposal has been the use of multi-view stereo, since everyone can have access to a camera. We have not explored more techniques.

2.3 Methodology and rigour

In order to develop the project we planned a rough development schedule that helped us in its creation.

First of all, we wanted to assure that the project was something that we were going to be able to do. Before deciding on choosing this project some basic tests were carried in order to decide if the project would be possible given the time and resource constraints. These basic tests consisted in doing some measures on a 3D model of a body part, these measures were closely related to the ones that we have needed in this project. They were successful.

The development of the project itself was started with the creation of a first version of the viewer. The reason to start with it is the fact that we needed the viewer in order to check any results, so it is the first we had to do, even if it has only basic features. It had to be able to load 3D models and do basic exploration tasks like pan, rotate or zoom.

The next step was exploring ways of comparing different states of the same body part. That involved the research and testing of different algorithms that give different measures that could be used for comparison. As it could be expected, in order to perform the tests, we needed at least a 3D model from a body part.

Finally, we evaluated other methods to obtain 3D models, apart from the light-based 3D scanner. That involved the use of the *Kinect* and the exploration of multi-view stereo techniques.

After having a basic version of the program which solved our problem, using at least a working comparison algorithm and a working scanning technique, we started making improvements to the program. It was an iterative process that involved the main program, the comparison algorithms and the scanning techniques.

2.3.1 Development tools

The development of the main program of our project has used *Qt* and *OpenGL*. *Qt* has been used for the creation of the user interface and *OpenGL* has been used for rendering 3D graphics. The main advantages of utilising them is that they are both open-source and are available in different platforms like *Windows*, *Linux* or *Mac*. 3D models have been modified with other tools like *Blender* or *MeshLab*, the first is essentially a 3D modelling program while the second provides tools to perform different transformations to 3D meshes. We have considered using external libraries, like the *Point Cloud Library*, since a lot of code that is related to 3D graphics has already been developed. The programming language has been *C++*, since it is the native language of the *Qt* package.

The process to obtain 3D models has needed additional tools, which depend on the device used to do this process. These tools will be presented when describing the scanning techniques that we are going to use.

The changes done to the code have been documented through a *BitBucket* repository, using the *Git* version control. The repository has been managed with a program called *SourceTree*, created by the same company that manages *BitBucket*.

The project documentation has been created with LaTeX, using *gnuplot* to generate plots, if needed.

2.3.2 Validation of results

To check if the program meets the objectives we have obtained a 3D model of a body part. Then, we have used a 3D model modification program like *Blender* or *MeshLab* to modify the model, simulating the effect of inflammation. Finally, we have used our program and the information it has given to decide if it is working properly. An alternative has been to compare a body part with the symmetrical one, in the case of using paired body parts.

Using the models obtained using the different scanning techniques in the process men-

tioned has been way to check that the scanning techniques that we have explored have been useful for our purposes.

Finally, it would have been desirable to use the program in a real case of inflammation. We would have liked to find a person that had inflammation on one of his joints to use the program to track the evolution of his or her inflammation, but it has not been possible. However, it is planned to show the tool to paediatric rheumatologists from Hospital Sant Joan de Déu, given that the MOVING research group of the Universitat Politècnica de Catalunya collaborates with it providing support in virtual reality applied to medicine.

2.4 Obstacles and risks of the project

The problem that we wanted to solve is well defined, and there are parts of the project like the creation of a 3D viewer or 3D scanning techniques that have been extensively developed. However, we knew that the development of the project was not going to be that easy, since we identified different potential problems. The main ones have been listed and explained in the following sections.

2.4.1 Main program

The main program is a 3D viewer with the features that this kind of application needs, that is, apart from navigating tools, which is something that every viewer has, there is a way to manage the different joints evaluated and the results of the comparison between joints are shown to the user as graphically as possible. No potential obstacles were found in the creation of the main program process since there are plenty of 3D viewers created, some of them being open source, so their codes could have been used as inspiration in case of difficulty. Also, we created similar programs in past courses. The main challenge that we have faced when creating the viewer is to create a program that is intuitive enough for someone that may not have any experience with 3D applications, but since usability is not a key area of the project, we preferred creating a functional program even if it is not easy to use for a non experienced user.

2.4.2 Comparison and analysis algorithms used

Although it seems easy to compare two states of the same body part and decide if the inflammation level has changed, it was potentially a difficult task to do. Having two models of the same body part, it is relatively easy to compare them if they are aligned and they have the same scale. But if the scale is different or they are not aligned, it is

more difficult. In the last case, we would need a way to align both models, which involves finding the rigid transformation, that is, the translation and the 3D rotation. Ideally, the application should be able to align both models automatically, with minimal or no user intervention. Moreover, not all the measures used to compare two models could have worked for our purposes. We have needed several tests to evaluate the usefulness of the different comparison techniques.

2.4.3 Scanning techniques

Current 3D scanners allow the rapid digitisation of 3D models with very high accuracy. Besides expensive 3D scanning equipment, we have also analysed the use of alternative low-cost techniques. *Microsoft*, creators of the *Xbox*, provide a SDK for the *Kinect*, and one of the features that it includes is the generation of a 3D model of a physical object. That means that potentially there should be no problems when using the *Kinect* to obtain a 3D model.

Multi-view stereo is more complicated. Since skin has got a homogeneous appearance, it could be difficult to match points in different images, hence making a precise reconstruction an impossible task.

Finally, having a 3D model is not enough to assure that it will work with our program. 3D scanning often generate models with undesirable properties caused by precision errors, noise or by the algorithm used in the construction of the model from the data gathered by the scanner. Resulting polygonal meshes often do not define the boundary of a valid solid, preventing us from computing volumetric properties such as volume and mass. These models need to be fixed in order to allow the comparison algorithms to work properly, but this is not always possible.

3 Contextualisation and bibliography

This section is about the context of the project and its bibliography. It features four different sections. The first one aims to describe the context of the project, that is, a brief description of the area of interest that the project is about and the actors affected. The second one gives information about the state of the art of the areas that we are going to explore in the project. The third one discusses whether to use or not the results found in the *State of the art* section. Finally, the fourth section presents the applicable laws and regulations.

3.1 Context

In this section we are going to do a description of the areas of interest that this project has been about, and we are going to explain which are the actors that have been affected by the development of this project.

3.1.1 Areas of interest

In this project we have had two clearly different areas of interest. The first one has been inflammation, which is what we are trying to help to track with the application resulting of the project. The second one has been the one which involves technical aspects of the project like the creation of the program or the obtaining of 3D models.

In the next lines we are going to detail both areas.

Inflammation

In this project we have made a program that is going to be used to track the evolution of inflammation. To be able to create a program that satisfied our needs, we had to know what inflammation is, taking into account its main characteristics.

Inflammation is part of the non-specific immune response that occurs in reaction to any type of injury. The five signs that characterise inflammation are pain, heat, redness,

swelling and loss of function, which can be explained by the changes caused to the body in the inflammatory process [1]. Inflammation can be caused by burns, infection by pathogens, physical injuries or by some diseases like asthma, allergies, inflammatory bowel diseases or rheumatoid arthritis.

In this project we have focused on the inflammation that appears in joints. It is a very common kind of inflammation, since it can appear as a result of an injury like a sprain or a luxation, which are both injuries that most of the people that practise any kind of sport will easily suffer. Also, there is a disease that causes it, rheumatoid arthritis, which affects between 0.5 and 1% of adults of the developed world [2].

We have already said that there are five signs that characterise inflammation. Its control is the way to track the evolution of inflammation. Our project has taken into account swelling only, since it is the sign that involves the geometry of the body part. Unlike medical imaging techniques, all 3D scanning techniques based on commodity equipment generate the 3D model of the body part that approximates the skin surface. The shape of the mesh depends thus, not only on the geometry of the underlying joint, but also on the different muscles and tissues beneath the skin. This means that, when shape differences are detected, we cannot distinguish whether the deviation is due to factors like fat concentration, muscle bulge, or joint swelling. Despite this inherent limitation of surface models, the availability of low-cost techniques for such a comparison is of great value, and has multiple applications beyond inflammation assessment, for example, estimation of muscle mass through comparison of 3D models in relax and contraction states, and assessment of the evolution of body shape and fat distribution. Nevertheless, the primary purpose of this project has been swelling evaluation.

Technical aspects

In order to develop the project, we have taken into account a few technical aspects.

First of all, we needed a program capable of representing a 3D model. That is, we needed a 3D viewer. We planned to include the common features a 3D viewer has, like rotate, pan and zoom, plus other features that could be useful for the kind of program that we are developing.

Also, we needed a way to obtain 3D models from physical objects. That is, we needed a 3D scanner. A 3D scanner is a device that analyses a real world object in order to obtain data from its shape, and possibly other features like the colour, and uses it to generate a 3D model which matches the attributes of the real object.

Finally, we needed methods to work with the 3D models so that we can get the data needed in order to give the user information about the evolution of his or her inflammation. That is, we need algorithms that are able to process 3D models in order to

obtain information from them. When trying to find algorithms that are able to process 3D models in order to give some data we entered a field called computational geometry. Computational geometry is a branch of computer science consisting of to the study of algorithms and data structures for geometric objects, with a focus on exact algorithms which are asymptotically fast [3].

In the *State of the art* section, we are going to give more details about the items mentioned in this section, giving then information about specific techniques that have already been developed.

3.1.2 Actors

The development of our project involves several actors, which are listed and described in the following lines.

Project developer

This project has had an only developer, myself. I have worked on everything which has been needed, that means that I am who has worked on the project plan, the documentation, the information research, the code and the tests.

Project director

The director of this project has been Carlos Andújar Gran, associate professor from the Computer Science department of Universitat Politècnica de Catalunya. His role has been to guide the developer of the project so that it can be carried it out when difficulties appear.

Users

Our aim has been to make a program such that everyone can be considered a potential user of it, so, if someone suffers from an injury that causes inflammation he or she could use our program to track its evolution. There is a problem, though, since there are scanning techniques that not everyone is going to be able to use. Nevertheless, we still consider everyone as a potential user of the project.

Benefited actors

Apart from the users, we can consider that doctors and physiotherapists are going to be benefited by the creation of our project, since using it their patients are going to be able to tell them about the evolution of their inflammation with precision, helping them to know if the treatment they are applying is effective.

3.2 State of the art

The goal of this section is to give information about the state of the art of the different elements that form our project. In the following lines we have listed the different areas that our project is going to explore, giving details of each of them.

3.2.1 Computing in medicine

Since computers started to became popular in the last century almost every field has been benefited by their evolution. Medicine has not been the exception, and that is the reason why doctors have been taking profit from the development of medical applications. Also, since smartphones have become popular, applications that help people to be healthy have been developed. In this section we are going to focus on the imaging techniques that have been developed in the last years to be used by doctors and we are going to show examples of applications that help people to be healthy.

Medical imaging

Medical imaging is the process of creating visual representations of the interior of a body in order to be analysed by medicine professionals. It aims to show the internal structures that are hidden by skin, bones and muscles in order to diagnose and treat diseases. It also allows professionals to identify abnormalities by making comparisons with data obtained from healthy patients.

The increasing popularity of computing has had also an effect in medical imaging with the creation of medical image computing. Medical image computing is a field that uses computer science, data science, physics, mathematics and medicine in order to develop computational and mathematical methods to solve problems related to medical imaging. Its The main goal is to extract clinically relevant information from medical images.

An example of the use of medical imaging combined with the use of a computer is computed tomography. A computed tomography (CT) scan is an imaging method that

uses x-rays to create pictures of cross-sections of the body. A CT scan creates detailed pictures of the body, including the brain, chest, spine, and abdomen. The test may be used to diagnose an infection, guide a surgeon to the right area during a biopsy, identify masses and tumours, including cancer and study blood vessels [4]. Figure 3.1 shows the image resulting of using this technique to evaluate damage on the brain of a patient.

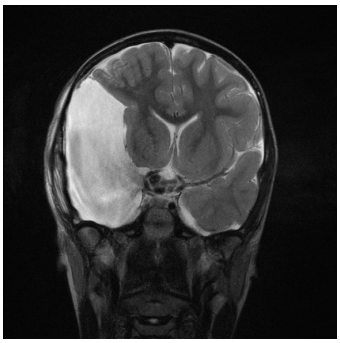

Figure 3.1: Computed tomography of the damaged brain of a patient.

Medical applications for patients

Although there are many tests that a patient can not do without having appropriate tools and knowledge, there are others that anyone can do. Here we are going to present an application that can be used to check a certain constant of the body.

An example of a medical application is *Instant Heart Rate*. This application, which is freely available for all the *iPhone* users, allows the user to find out his or her heart rate in a few seconds just placing the tip of one of his or her fingers on the *iPhone* camera. It is continuously rated as the world's best mobile heart rate measurement app and is trusted by Stanford's leading cardiologists for use in clinical trials [5].

3.2.2 3D viewers

We can define a 3D viewer as a program capable of loading 3D models in order to show them to the user. With that definition, every program with a 3D window includes a 3D viewer, so, all the programs used in 3D modelling and all the 3D-based video games are examples of 3D viewers. Knowing the huge amount of modellers and video games that exist, we can say that this field has been deeply explored.

In this section we are not going to present different 3D viewers. We are going to present the most used frameworks that allow rendering of 3D graphics, since each 3D viewer makes use of at least one of them. These frameworks are *OpenGL* and *Direct3D*. Nowadays both are extensively used.

OpenGL

OpenGL is the main environment for developing 2D and 3D graphics applications. Since its introduction in 1992, *OpenGL* has become the most used and supported 2D and 3D graphics framework in the industry, bringing thousands of applications to a wide variety of computer platforms.

Any visual computing application requiring maximum performance can make extensive use of the high quality and high performance *OpenGL* capabilities, which allow developers in diverse areas, such as broadcasting, entertainment, medical imaging, CAD and virtual reality to generate and display top quality 2D and 3D graphics.

The most important features of *OpenGL* are the following [6]:

- It is an industry standard, being mantained by an independent consortium, the *OpenGL Architecture Review Board*.
- It is stable, and its backward compatibility ensures that existing applications do not become obsolete.
- It is portable, since all *OpenGL* applications generate consistent visual display results on any hardware, regardless of the underlying software.
- It is available in a great amount of platforms, such as *Linux*, *Windows* or *Mac*.
- It is always evolving, with the new hardware capabilities being added through extensions.
- It is scalable, since *OpenGL* based applications can run on all kinds of systems, from mobile devices to supercomputers.
- It is easy to use, since *OpenGL* is well structured with an intuitive design and logical commands.
- It is well documented, given the amount of books and sample codes available.

There have been several versions released, the latest being the 4.5 version. Each version includes new extensions that add new features, and using the newer versions is not a matter of software only, since newer versions can be run on supported hardware only.

OpenGL processes the primitives through a pipeline that receives data from vertices as input and does operations on them, generating an array of pixels which is stored in a frame buffer. In the first versions of *OpenGL* this pipeline was fixed, but in the last years it has become programmable, meaning that custom code can be created for some stages. These programs are called shaders, and there are several kinds of them, such as vertex shaders, geometry shaders or fragment shaders, depending on the stage that they

are replacing.

GLSL is the language used to code shaders. It was introduced in the 2.0 version of *OpenGL* and it is a language with a syntax based on *C* that includes plenty of built in functions and data types such as trigonometric functions or vectors and matrices to allow the programmer to transform vertices or fragments.

Direct3D

Direct3D is a 3D graphics framework created by *Microsoft* to be used in its platforms, such as *Windows* or the *Xbox* console. It is part of the *DirectX* framework, which includes full multimedia support, meaning that not only graphics are covered but also audio.

Direct3D is a low level framework that can be used to draw 3D primitives such as triangles, lines or points, or to start highly parallel operations on the GPU. Its aim is to allow the user to design code with a high level of abstraction from the GPU implementation.

There have been several *DirectX* versions released, including different versions of *Direct3D*, starting from the first one, released in 1995. The 9 version is the oldest that is still supported, and it is has to be used it in machines using *Windows XP* and newer *Windows* versions, while the 10 version has to be used in *Windows Vista* and newer *Windows* versions and the 10.1 and 11 versions have to be used in machines running *Windows 7* or *Windows 8*. Not only the software has to be adapted in order to use different versions of *Direct3D*, since all hardware supports the 9 version but for newer versions the hardware has to be capable of supporting it [7].

Direct3D is used in many *Windows* applications that make use of 3D graphics, and it is widely used in computer games developed for *Windows*.

Direct3D processes primitives through a pipeline that uses data from vertices as input and generates an array of pixels which is stored in a buffer. In the latest versions some stages of the pipeline can be programmed, creating shaders such as vertex shaders or pixel shaders, depending on the stage that the code is used in.

HLSL is the language used to create shaders. It was first introduced in the 9 version of *Direct3D*.

3.2.3 Algorithms to compare 3D models

In this section we are going to present different algorithms that we have planned to use in order to fulfil the requirements of our project.

The first we have to do in order to compare two 3D models that represent the same physical object is to align them. The process of finding a transformation to align two meshes or point clouds is also called registration.

In order to perform the registration process, we present an algorithm called Iterative Closest Point (ICP). The ICP algorithm has become the dominant method for aligning three dimensional models based purely on the geometry, and sometimes colour, of the meshes. ICP starts with two meshes and an initial guess for their relative rigid-body transform, and iteratively refines the transform by repeatedly generating pairs of corresponding points on the meshes and minimising an error metric [8]. In our case, a rigid body transform is not going to match all points, since the 3D models used are not going to have exactly the same shape, but it should be working doing slight modifications to the algorithm if needed.

Once we have the different meshes aligned we have to use a way to measure its differences. Basic ideas include measuring areas or volumes, having algorithms that we consider too basic to be included in this document. A more complex idea consists on using a structure called Reeb graph. A Reeb graph is an interesting graph to describe topology structure that encodes the connectivity of its level sets based on the critical points of a scalar function [9]. We can think about the Reeb graph as a graph that represents a "skeleton" of the mesh, thus containing information about its shape. Figure 3.2 shows an example of Reeb graph.

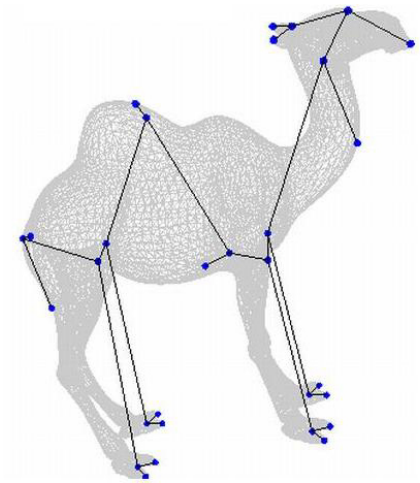

Figure 3.2: Reeb graph of a 3D model of a camel.

Another structure that it can be used to summarise the information about the geometry of the mesh in a simplified way is the medial axis. The medial axis of a solid is defined as the set of centres of maximal balls contained in the solid. It has been proposed as a tool for shape analysis, surface reconstruction, motion planning, and many other applications. It is useful because it provides a local lower-dimensional characterisation of the solid. In particular, for a solid in 3D the medial axis consists of a union of surfaces that provide information about the shape and topology of the solid. If the distance to the boundary is also stored for each medial axis point, the resulting structure is known as the medial axis transform (MAT) and the entire boundary representation can be reconstructed from it [10].

Finally, we can measure distances with the calculation of a distance field in the object. Given a set of objects, a distance field in 3D is defined at each point by the smallest distance from the point to the objects. Distance fields are frequently used in computer graphics, geometric modelling, robotics and scientific visualisation [11]. Computation of distance fields is closely related to the medial axis since some implementations use a distance field to calculate the medial axis [10]. Figure 3.3 shows an example of distance field.

Figure 3.3: Distance field that shows the distance to the surface of a 3D model.

Since techniques to process 3D models have been extensively developed, we can find programs and libraries that can be useful to check if the algorithms give us the desired results, or that can be used in our project so we do not have to implement the algorithms by ourselves. We are going to mention two of them, *MeshLab* and *Point Cloud Library*.

MeshLab

MeshLab is an open source tool that aims to process and edit 3D meshes, focusing on the tasks that models generated by a 3D scanning process usually need to be done on them

[12]. It can be useful when trying to check if the algorithms that we have mentioned in this section can provide the results that we desire, since it allows the user to align different 3D models that represent the same object using the ICP algorithm, and it also can calculate the distance between two 3D models, creating a distance field. Moreover, it contains plenty of tools to work with 3D models that, for example, allow performing computations of different measures such as the area or the volume, or modifying a mesh by applying a transformation on it, simplifying it or deleting the parts that are not useful.

Point Cloud Library

Point Cloud Library is a library whose main objective is to process 3D images and point clouds using state of the art algorithms [13]. Since the set of vertices of a 3D mesh can be considered a point cloud, the algorithms that this library provides can be used in our project. It contains classes to perform operations of filtering, segmentation or registration, the last one including the ICP algorithm. It also provides code to find features and key points and to generate meshes from point clouds. Moreover, it contains data structures that optimise working with large point clouds and classes to allow the programs created to perform input and output operations and to have a 3D window showing the point cloud.

3.2.4 3D scanning techniques

In this section we are going to describe the 3D scanning techniques that we aim to explore in order to know what are the results that have been achieved using them.

Light-based 3D scanning

A structured-light 3D scanner is a 3D scanning device that gathers data from the shape of a real world object by projecting a light patterns on the object.

The research group on modelling, visualisation, interaction and virtual reality of Universitat Politècnica de Catalunya, called MOVING, has a research laboratory called Centre de Realitat Virtual, and among all its equipment there is a light-based 3D scanner. It is a scanner from *Artec 3D Scanners*, namely, the *Artec MHT* scanner, and its resolution is up to 0.5 mm [14], which gives us more than enough precision to solve our problem. *Artec 3D Scanners* provide software capable of processing the data obtained by the scanner in order to create a 3D mesh that we can use for our purposes. The scanner is shown in figure 3.4.

Figure 3.4: *Artec MHT* 3D scanner.

In the initial stages of the project we did a test with this scanner, scanning both knees of a person. The result was a high-resolution mesh that fulfilled all the requirements that we needed in order to use it as a test case for our project.

Kinect

A *Kinect* is a device created by *Microsoft* that contains different sensors like a microphone, a colour camera and a depth camera in order to allow the user to interact with a computer using gestures or the voice. Figure 3.5 shows a *Kinect* device.

Figure 3.5: *Kinect* for *Windows* device.

There have been two versions of the *Kinect* sensor released, the *Xbox 360* version and the *Xbox One* version. Their specifications are different, with the latter being better and having as most remarkable change the improvement of the precision of the depth camera.

When most of the people think about a *Kinect*, it gets associated with gaming. It is partially true, since it has become popular thanks to it being sold with the *Xbox*, but it has other uses. A version for *Windows* was released for each of the *Kinect* sensors mentioned. This version enables the development of applications that allow people to interact naturally with computers by simply gesturing and speaking. There are examples of *Kinect* for *Windows* being used in health care or in education [15].

Microsoft provides a SDK to let developers design programs to work with the *Windows* version of the *Kinect*. One of the tools included in the SDK is *Kinect Fusion*, which is a program that provides 3D object scanning and model creation using a *Kinect* for *Windows* sensor. The user can paint a scene with the *Kinect* camera and simultaneously see, and interact with, a detailed 3D model of the scene [16]. This process is done using the depth camera that *Kinect* has built in. It has to be noted that the SDK is also working for the *Xbox* versions of the *Kinect*, so it can be used to obtain 3D models too.

While the results are not as good as what it can be obtained with a light-based 3D scanner, the models obtained can be used in our program, and taking into account the difference in price between the devices, we thought that it was worth exploring the possibility of using it in the project.

Multi-view stereo

The goal of multi-view stereo is to reconstruct a complete 3D object model from a collection of images taken from known camera viewpoints.

The most important part in the scanning through this technique are the algorithms used to reconstruct the mesh. There are many algorithms for this purpose, some of them have been analysed and we can find algorithms that are able to reconstruct a 3D model with great accuracy [17].

A problem arises, though, since doing a reconstruction using images can be done when there are enough features to allow the algorithms to match the different images. In our problem, we have tried to reconstruct a body part, which usually has a homogeneous colouring, so we knew that it could be difficult to get any multi-view stereo technique working, because it may be difficult to create a 3D mesh even for a human.

Autodesk have created an application called *123D Catch* that allows users to load photographs of an object in order to create a 3D model, which could be printed using a 3D printer. In their website we can find examples of captured objects using this technique, and if the photographs are taken following the guidelines they provide, the result is pretty accurate [18].

Figure 3.6 shows an example of use of *123D Catch*.

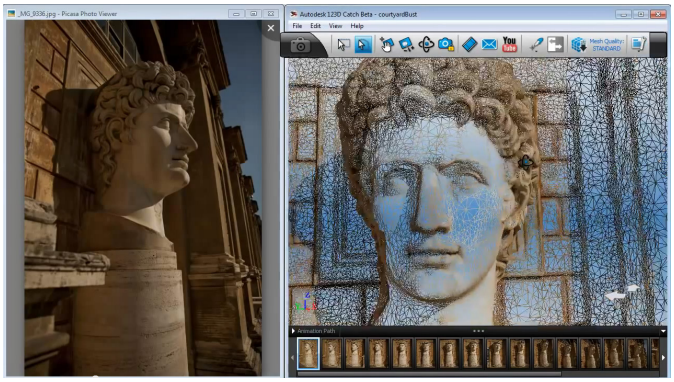

Figure 3.6: *123D Catch* in action. In the left, picture of the object. In the right, 3D model reconstructed using pictures.

3.3 Use of previous results

In this section we are going to discuss whether we should use the solutions that we have presented in the *State of the art* section or not in each of the areas explored.

3.3.1 3D viewers

Our idea has been to create our own 3D viewer, and it was clear that we were going to use any of the existing frameworks that allow developing applications that generate 3D graphics. We have presented two of these frameworks, *OpenGL* and *Direct3D*, which are the most used ones. We decided on using *OpenGL* because it is cross platform, unlike *Direct3D*, which is available for *Windows* only. Also, we already had experience working with it, which does not happen for *Direct3D*. Furthermore, the *Qt* platform has support for *OpenGL* graphics, so we could use it to create our viewer.

3.3.2 Algorithms to compare 3D models

We have presented several algorithms that can help us to compare different 3D models. Since we did not know if they were going to be useful for our purposes, we decided on testing different algorithms in order to check whether they are useful or not, and make possible to offer the user alternatives.

We have introduced software that can be used in this area. Since it is usually a good practice to use existing code, we have used *MeshLab* for testing purposes and the *Point Cloud Library* in the final implementation when using algorithms that *MeshLab* includes.

3.3.3 3D scanning techniques

We have presented three different 3D scanning techniques. We planned to use the light-based 3D scanning technique that the 3D scanner from Centre de Realitat Virtual provides. Also, we planned to use *Kinect Fusion* in order to use a *Kinect* to do 3D scanning. About multi-view stereo, we have used *123D Catch*, although we first planned exploring different multi-view stereo techniques.

3.4 Applicable laws and regulations

In this section we are going to comment which are the applicable laws and regulations that we have to take into account in the development process of this project. It has been divided in two parts, the first one being about the software licenses used and the last one being about the applicable laws that affect our project.

3.4.1 Software licenses

The licenses of the software that we are going to use in the project, which has been listed in the *Methodology and rigour* section, are the following.

- *Qt*: GNU General Public License (version 3) and the GNU Lesser General Public License (version 2.1).
- *OpenGL*: Free Software License based on BSD, X, and Mozilla licenses.
- *Point Cloud Library*: BSD license.
- *Blender*: GNU General Public License.
- *MeshLab*: GNU General Public License (version 2.0).
- *Git*: GNU General Public License (version 2.0).
- *SourceTree*: Proprietary license.
- *Kinect SDK*: Proprietary license.
- *123D Catch*: Proprietary license.
- LaTeX: LaTeX Project Public License (version 1.3c).
- *gnuplot*: Open source license.

As it can be observed, we have made extensive use of free software. Doing that we reduce the cost of the project without losing quality, and it allows everyone to be able to use the software generated or even improve it.

3.4.2 Laws and regulations

While there are projects that are about areas that are subject to strict regulations, we have not found any other law or regulation that may be considered in order to develop this project legally.

4 Temporal planning

This section is about temporal planning, and it aims to describe the tasks had to be executed in order to do the project, giving an action plan that summarises the actions that had to be taken in order to finish the project in the desired time frame. However, we have to take into account that the planning described in this section is the initial planning and it had to be modified due to the development pace of the project. The modifications that it suffered have also been described.

The project was started on July 1st, 2014 and its deadline has been January 23th, 2015. Therefore we have to develop our schedule given these time constraints.

4.1 Description of tasks

In this section we are going to describe the tasks that we had planned to do in order to make our project.

4.1.1 Viability tests

This was the first task that we had to do. It was done before deciding on this project, since it was used to know if the project was even possible.

It consisted in developing an algorithm which had a 3D model as input, using it to do a voxelisation of the space to calculate the distance of each voxel to the boundary of the object represented by the model. We already had some parts of the algorithm and some 3D related code so what we had to do was to generate a code that could be used for our purposes.

The tests were successful, that is the reason that led us to choose this project.

This task did not take much time if we compare it with the following tasks. After completing it, a long break was taken, reason being holidays and having a job which lasted for nearly one and a half months.

4.1.2 Project planning

This task was the first task that had to be done after the viability tests since it defines what was going to be done in the following tasks and the way it was going to be done. It was divided in the following four stages:

- Scope of the project.
- Temporal planning.
- Economic management and sustainability.
- State of the art.

As a result, a document was created, which has been the base of the initial sections of this document.

4.1.3 Initial system set up

Before starting the development of the project itself, we needed to set up the tools required to work on it. It may seem a simple operation, but it took enough time to be listed as an important task in our schedule.

To develop the project, we needed the *Qt* platform to create the main program, the *Point Cloud Library* in order to have a library containing algorithms and structures implemented so we could use them, *MeshLab* and *Blender* as tools used to modify 3D models in the case of needing to do it and a platform able to compile LaTeX code and create plots with *gnuplot* to generate documentation. We wanted to test the program in *Windows* and *Linux*, so we needed to install both operating systems and the needed software in both of them.

After installing all these software, configuring it and checking that everything worked as we expected, we could start working on the most important phase of our project.

4.1.4 Main development

This was the main task of the project. It covered all the tasks related with implementation and testing of the program that solves the problem stated in the *Scope of the project* section. It was divided in the following stages:

- Viewer programming: The viewer was the first thing that we had to implement since it was the program that we had to use to test the algorithms that we developed

in the following stages. We have coded a *Qt* application capable of loading 3D models and navigate through them. It has been designed so it becomes an easy task to use the algorithms found in the following stages.

- Comparison and analysis algorithms programming: This phase was divided in three stages: research, programming and testing. The first stage was about doing some research in order to find algorithms that could be used to compare 3D models that represent the same physical object. We knew that these kind of techniques already existed, so we had to look through the different algorithms to select the most useful ones. In the second stage we implemented the algorithms found in the previous stage so that they worked with our program. That involved coding the algorithm, or creating code to use a working implementation of it, and modifying the viewer so it could be used to check the results of its execution. After having the algorithms implemented, we entered the third stage, in order to test that they worked for our purposes. In order to do it, we used a 3D model of a body part and with *MeshLab* or *Blender* we modified it, simulating the effects of inflammation in the geometry of the body part. We have generated plots with *gnuplot* to easily evaluate the results of the algorithms.

- *Kinect* scanning: This phase consisted in doing some research in order to be able to use the *Kinect* device to get a 3D model from a physical object. The research started with the resources that *Microsoft*, creators of the *Kinect*, provide, but we wanted it to become more extensive.

- Multi-view stereo scanning: This seemed to be the most difficult stage of the project and the one that we would have avoided doing or would have simplified in the case that the other stages would have taken more time than we expected. It initially consisted in doing research among the multi-view stereo related bibliography in order to be able to implement a multi-view stereo method that can be used to generate a 3D model from a body part using several pictures of it.

- Scanning tests: In this stage we were going to use the 3D models generated by the light-based scanner, the *Kinect* scanning and the multi-view stereo methods in our program. If the results were not positive, we would need to do more research.

The dependencies between the stages mentioned are easy to describe. First of all, the scanning research stages had no dependencies. We already had a working technique, the light-based scanner and we knew that we could use any 3D model from a body part, even the ones which were contained in on-line model repositories, so there was no problem in doing the scanning research related stages at the end or at the beginning. Apart from that there were only two restrictions: the viewer had to be done before testing algorithms and the scanning tests had to be done after having a working algorithm.

Also, we had to take into account that the program and any of the components that were part of it could have been improved before the delivery of the project, so any of

the tasks described before could have been done more than once if needed.

Finally, it is important to note that for each stage we wrote its documentation during the stage itself since we thought that it was better to document the project while it was on development.

4.1.5 Final task

In this task we checked that everything worked as expected and we prepared the delivery of the project, assuring that the documentation was correct and preparing the final presentation.

4.2 Time table

The table 4.1 summarises the time spent in each of the tasks described in the previous section.

Task	Time spent (hours)
Viability tests	20
Project planning	80
Initial system set up	5
Viewer programming	60
Comparison and analysis algorithms programming	150
Kinect scanning	50
Multi-view stereo scanning	75
Scanning tests	25
Final task	35
Total	500

Table 4.1: Summary of the time spent in each task.

4.3 Resources used

We can divide the resources that we planned to use in hardware resources and software resources. Here are both lists, with the tasks that each resource is used in included.

4.3.1 Hardware resources

- PC (with the following specifications: Intel Core 2 Quad Q9550 at 2.83 GHz, 4 GB RAM, nVidia GeForce GTX 650 Ti Boost): used in all tasks of the project.
- *Kinect*: Used in *Kinect* scanning and scanning tests
- Light-based scanner: Used in scanning tests.
- Camera: Used in multi-view stereo scanning and scanning tests.

4.3.2 Software resources

- *Windows* 7: used in all tasks.
- *Ubuntu* 13.10: used in all tasks.
- *SourceTree*: used in all tasks.
- *Qt*: used in viewer programming.
- *Point Cloud Library*: used in comparison and analysis algorithms programming.
- *Blender*: used in comparison and analysis algorithms programming.
- *MeshLab*: used in comparison and analysis algorithms programming.
- *Kinect SDK*: used in *Kinect* scanning.
- LaTeX: used in all tasks.
- *gnuplot*: used in comparison and analysis algorithms programming.

4.4 Gantt chart

Our initial schedule is represented by the Gantt chart shown in the figure 4.1. We took into account the break for job and holidays mentioned before.

	Name	Length	Start	End	Predecessors
1	Viability tests	15d	01/07/2014	15/07/2014	
2	Job	45d	16/07/2014	29/08/2014	
3	Holidays	9d	30/08/2014	07/09/2014	
4	**Project planning**	**25d**	**08/09/2014**	**03/10/2014**	**1**
5	Scope of the project	4d	08/09/2014	12/09/2014	
6	Temporal planning	4d	13/09/2014	17/09/2014	5
7	Economic management and sustainability	4d	18/09/2014	22/09/2014	6
8	First presentation	5d	23/09/2014	28/09/2014	7
9	State of the art	4d	29/09/2014	03/10/2014	8
10	Initial system set up	1d	04/10/2014	04/10/2014	4
11	Viewer programming	19d	05/10/2014	23/10/2014	10
12	Algorithms programming	42d	24/10/2014	04/12/2014	11
13	Kinect scanning	14d	05/12/2014	18/12/2014	
14	Multi-view stereo scanning	21d	19/12/2014	08/01/2015	
15	Scanning tests	7d	09/01/2015	15/01/2015	13,14
16	Final task	8d	16/01/2015	23/01/2015	12,15

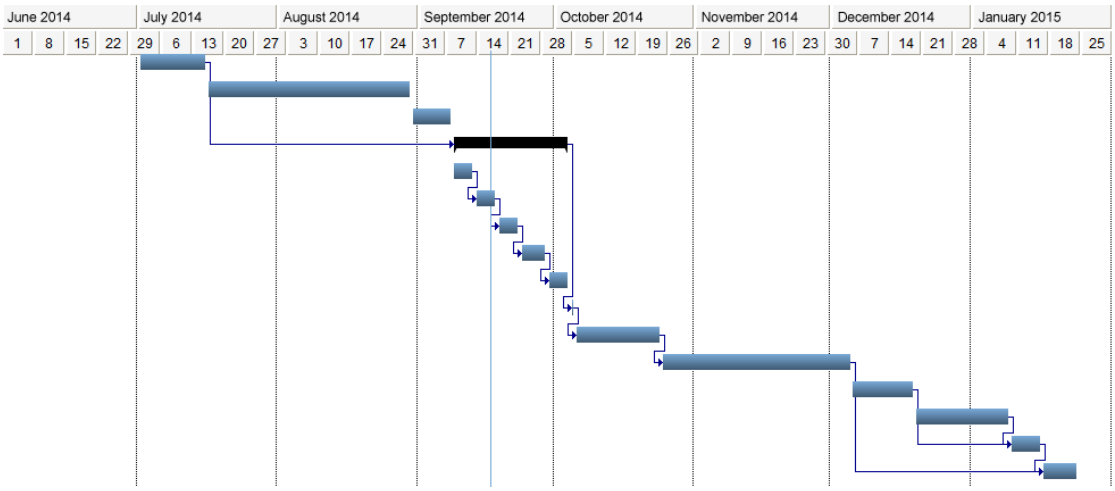

Figure 4.1: Gantt chart of the project.

4.5 Action plan

In this section we are going to describe how we planned to execute the plan we have created.

Our idea was to work as the schedule says, executing the tasks in the order stated in the Gantt chart, but we knew that like in every project there would appear obstacles that would make it difficult to follow the plan. We had to take into account that the time was limited, and there are things that were a must. We tried to do all the tasks stated but if on any of them we had ran out of time we would have decided on doing a basic version only. Of course, there were priorities. We needed to have a viewer and

we needed to have a working algorithm, and it would have been very interesting to have a scanning technique apart from the light-based 3D scanner. We knew that multi-view stereo was something that would be very interesting to use but we knew that it seemed difficult to make it work, at least, using our own code. So what we did was to make sure that a working viewer and a working algorithm was completed in the time that we expected it to be completed. In case of needing more time, we were going to take it from the time assigned to the scanning techniques, trying to have enough time for the *Kinect* research.

We tried to arrange meetings with the project director every time that an important stage of the project is finished.

From the time table it can be deduced that we needed a total time of 500 hours. Considering that we had approximately 20 weeks to develop the project, that is 25 hours per week, which made it possible to finish the project in the given period.

4.6 Modifications caused by the development process

During the development of the project we found that the planning could not be strictly followed, so we had to do some changes to adapt it to the circumstances that we had to face in the development process and make it viable.

The most important change came from the fact that the *Viewer programming* task required more time, since every time that we wanted to add a new feature, the viewer had to be modified so that it could show the user the new feature, and during the proposal of features that we did while exploring algorithms to help in the comparison process it had to be modified several times. It can be deduced from this fact that the *Viewer programming* task and the *Comparison and analysis algorithms programming* task had to be interleaved. Using more time in improving the viewer meant less time for other tasks, since the deadline was still the same. The time spent in the scanning tasks was reduced, since scanning with the *Kinect* was straightforward, knowing that *Microsoft* provided software to use the *Kinect* as a 3D scanner, and we needed time to set up the system only. Also, the use of multi-view stereo algorithms was marked as optional in the initial planning, but we still did it in a small amount of time, since we made use of *123D Catch*, while in the initial planning we wanted to explore different techniques and even implement some of them by ourselves.

5 Budget and sustainability

This section is about the budget and the sustainability of the project. For this reason, it contains a detailed description of the costs of the project, describing both material and human costs, an analysis of how the different obstacles could have affected our budget and an evaluation of the sustainability of the project.

5.1 Budget estimation

In this section an initial estimation of the budget needed in order to make our project possible according to the initial planning is shown. We have divided the budget in three sections, depending on the kind of resources that we are taking into account. They are hardware, software and human resources. At the end of the section, we have shown the total budget obtained combining the three sections mentioned.

To calculate the amortisation we have taken into account two factors, the first one being the useful life and the second one being the fact that our project has lasted for approximately six months.

5.1.1 Hardware resources

The table 5.1 contains the costs of the hardware that we have used in the development of the project.

Product	Price	Useful life	Amortisation
PC (including all the needed devices)	1000.00 €	4 years	125.00 €
Light-based 3D scanner	15000.00 €	4 years	1875.00 €
Kinect	200.00 €	4 years	25.00 €
Photograph camera	100.00 €	4 years	12.50 €
Total	16300.00 €		2037.50 €

Table 5.1: Hardware budget.

5.1.2 Software resources

The table 5.2 summarises the costs of the software that we have needed to develop our project.

Product	Price	Useful life	Amortisation
Windows 7 Professional	130.00 €	3 years	21.67 €
Ubuntu 13.10	0.00 €	3 years	0.00 €
SourceTree	0.00 €	3 years	0.00 €
Qt	0.00 €	3 years	0.00 €
Point Cloud Library	0.00 €	3 years	0.00 €
Blender	0.00 €	3 years	0.00 €
MeshLab	0.00 €	3 years	0.00 €
Kinect SDK	0.00 €	3 years	0.00 €
LaTeX	0.00 €	3 years	0.00 €
gnuplot	0.00 €	3 years	0.00 €
Total	130.00 €		21.67 €

Table 5.2: Software budget.

5.1.3 Human resources

The table 5.3 shows the costs of the human resources needed in the development of our project.

Role	Price per hour	Time	Cost
Project manager	50.00 €	80 h	4000.00 €
Software designer	35.00 €	160 h	5600.00 €
Software programmer	25.00 €	135 h	3375.00 €
Software tester	20.00 €	125 h	2500.00 €
Total		500 h	15475.00 €

Table 5.3: Human resources budget.

We have detailed how the different tasks have been divided, so it is easier to understand the number of hours given to each role stated in table 5.3.

The project manager has worked on the project planning only.

The software designer has worked on everything that involves programming, that is, the viability tests, the viewer programming and the comparison and analysis algorithms

programming. Also, *Kinect* scanning and multi-view stereo scanning may involve programming. The same can be applied to the software programmer. The software tester has taken part in the programming tasks, since it has been useful to do some tests to check the progress, but his or her main tasks have been the scanning tests and the final task, which involved extensive testing. Taking into account that, knowing the number of hours that were spent in each task and making a rough calculation using the participation of each role in each task that gives the number of hours stated in table 5.3.

5.1.4 Total budget

Using data shown in tables 5.1, 5.2 and 5.3, we can use table 5.4 to describe the total cost of the project.

Concept	Cost
Hardware resources	2037.50 €
Software resources	21.67 €
Human resources	15475.00 €
Total	17534.17 €

Table 5.4: Total budget.

5.2 Budget control

Our budget could have needed a modification if the development of the project would have made it impossible to follow the plan established, which is something that is always likely to happen.

We could have had difficulties in our project, but it was unlikely that we would need any hardware resource apart from the resources that have been already listed in the budget estimations shown in the previous section. We might have needed more software resources, but nowadays there are plenty of free applications that can act as a substitute of almost any commercial application. So, the budget section that we have to control is the one related to the human resources.

As stated in the *Temporal planning* section, the task that may have got longer is the programming of algorithms to compare 3D models. This task involves software designers, programmers and testers, so we have to take into account that the money spent on paying them could have grown if the problem had become more difficult than we expected when doing our plan.

5.3 Sustainability

In this section we have evaluated the sustainability of our project in three different areas: economic area, social area and environmental area.

5.3.1 Economic sustainability

In this document we already can find an assessment of the costs of our project, taking into account both material and human resources.

The cost stated in the *Budget estimation* section of this document could have been the only cost spent in the project, since we aimed to create a working program, so it will not need maintenance in the case that we do not create any update. However, it is possible that further development is needed. Creating an improved version of the program with more features or adapting it to work in any new platform that is released in the future would add more cost to its development.

It would be difficult to do a similar project with a lower cost. It is difficult to say that we could have done it in less time, so the money spent in human resources can not be less, same happens with software, everything is free except the *Windows* operating system, and if we wanted to test it in *Windows*, we needed to buy it. We could have spent less money on hardware, but that could imply worse results. For example, we could have avoided using the light-based 3D scanner, but then we would lose a reliable tool to get scanned 3D models. What we could have done is to rent the scanner every time we needed it, so the price would have been much lower.

We can not assure that the cost of the project would make it competitive, but we can say that it depends on the distribution method. We could make it a free application, so it would be very competitive, but then we would need a way (for example, advertisements) to get paid every time that the application is downloaded so we could recover the money invested. In the other hand, we could make it a paid application, but its price should not be too high.

We have spent most of the time in the tasks which are more important. As the *Temporal planning* section shows, most of the time is going to be spent in trying to find comparison algorithms, and we consider that being the most important task of our project since the quality of the algorithms determines the quality of our program.

It is not the case, but is important to remark that this project could have been executed with collaboration of medical institutions, since they may be interested in developing an application like that one.

5.3.2 Social sustainability

The application that we have developed in this project is going to be used in the countries which are developed enough so that everyone can have access to a computer, since our goal is that it can be used by everyone who can have access to a device capable of doing a 3D scan.

Everyone with an injury or suffering a disease that causes inflammation is going to be able to use the program resulting of the development of our project in order to track the evolution of his or her inflammation. The program gives the patient a way to measure inflammation, before its creation he or she had to rely on its memory to determine if the swelling level increased or decreased.

It may seem that doctors are going to be affected negatively by our project, but we think that it is not the case. We want our application to become a tool to help a person to track the evolution of inflammation, but if he or she wants to have good advice about how to heal his or her injury or how to treat any of the inflammatory disorders that he or she is suffering, he or she has to go to a doctor.

5.3.3 Environmental sustainability

The resources used in the project have been detailed in the *Budget estimation* this document and in the *Temporal planning* section.

During all the development of our project we have had a computer running. Also, during some phases we have used a light-based 3D scanner, a *Kinect* or a camera. The devices used, the energy spent and the paper used to print the documentation have been the only resources used.

Knowing that, we can estimate the energy spent developing the project. We can suppose that our computer consumes an average of 300 W when we are working in the project. Since we have needed to use the computer in all the development of the project, which is 500 hours long, the energy we need is 150 kWh, which is equivalent to 57.75 kg of CO_2. It is in fact a high amount of energy, but since we have needed to make extensive use of a computer there is no way to reduce it.

We could have used more code from other projects. For example, there are already libraries that offer code to create a viewer or to import 3D models. This would imply spending less time in the project and a consequence is that less energy would be needed, but we preferred to develop most of the features by themselves.

5.4 Modifications caused by the development process

In the *Temporal planning* section we have seen that the initial planning has needed a modification in order to adapt it to the real development pace. It can be deduced from this fact that the budget of the project may need a modification to adapt it to the new planning. This is what we are going to comment in the next lines.

The software and hardware used has not changed, with the exception of the addition of *123D Catch*, which is free, so the only area of the budget that is subject to modification has been the human resources budget. However, since the deadline of the project has not changed and the tasks that have become longer are replacing other tasks and the distribution of roles among these tasks is almost the same, that is, the software designer makes an initial design, then the software programmer codes it and finally the software tester does the testing phase, the budget after this modification is roughly the same.

6 Software design overview

In this section we are going to describe briefly the process that we followed in order to develop the software that solves the problem proposed in the *Scope of the project* section.

We have created a program that contains all the features needed to compare data of the same body part in different states. So, no additional software is necessary except the applications needed to obtain data from the body parts that are going to be processed.

The first that we had to do was to set up our system, installing all the software needed in order to start the development. The setting up of the software was a very simple process, but it took a few hours, time that we could not have avoided spending. We had to download the required installers, which are usually found in the official websites, install them and check that everything worked as expected. Once it was done, we started with the development.

The creation of this program has been divided in three different stages:

- Viewer design and implementation: In this stage we have created a 3D viewer, which is the program that we have presented as the result of the project. It has a 3D window with navigation features implemented, and it can load 3D models in order to show them in the 3D window. It has been created using the *Qt* framework and it had to allow the code developed in the next two stages to be added.

- Comparison and analysis algorithms exploration and implementation: In this stage we have explored different algorithms to align and compare 3D models. We did some tests first, using *MeshLab*, in order to check if the algorithms included on it were useful for our purposes. Then, we did some tests with the *Point Cloud Library*, which is what have used in our program, in order to understand how does it work. After that, we added new code to our program which makes use of this library. Once we had done that, we did more testing to decide the parameters of the algorithm which work best, making extensive use of plots. Finally, we coded the algorithms that the *Point Cloud Library* did not have implemented.

- Scanning techniques exploration and implementation: In this stage we have explored different techniques that allow us to acquire 3D models, implementing code if the codes that had already been developed were not enough to fulfil our requirements.

We have used *Windows 7* as the operating system to develop the project. Almost all the tools needed to develop the project are available for both *Windows* and *Linux*, but there is a set of tools, the *Kinect SDK*, which is available for *Windows* only. Nevertheless, since all the libraries that we have used in our code are cross platform, it will be able to be compiled in a *Linux* platform.

The following three sections contain a detailed description of the development process of each of the stages that we have mentioned previously. The results have been summarised using figure 6.1.

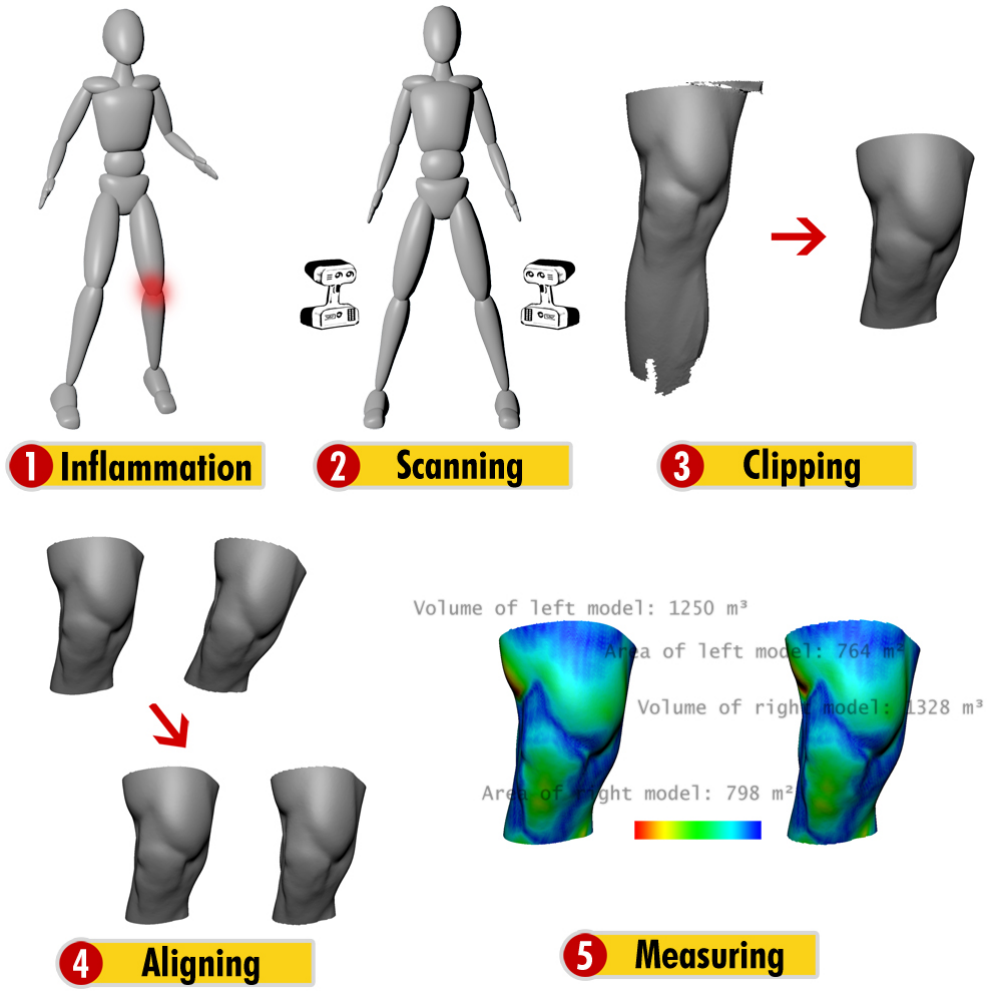

Figure 6.1: Summary of the usage of our application.

7 Viewer

In this section we have described the process of planning and implementing the main program of our project, which, as told before, is an enhanced version of a 3D viewer.

7.1 Design plan

Our aim has been to create an application which when running shows a window with the standard format of the operating system. That means that it has to contain a menu bar containing menus with the options that the user can choose and a status bar to show extra information, and it has to show dialogues when needed. The most important feature of this window is the fact that it contains an *OpenGL* window which covers most of its space.

We have included options to load 3D models, change the background colour, reset the navigation transformations, added to the standard options of a window like about or exit. More options were added when coding features related to the comparison of different models, since a menu option is the way to let the user execute any feature.

About the *OpenGL* window, it shows the 3D models that have been loaded, but there are different viewing modes. The user has the ability to choose if he or she wants it to be divided in two halves in order to show two models at the same time in order to do a visual comparison, or if he or she wants only one of the two models to be shown.

We have used the *Qt* framework to create this application since it is very useful when it comes to creating a GUI. *Qt* code uses the *C++* language with its own classes and other extensions, so there are some concepts that have to be defined in order to do a proper description of the developing process.

First of all, we have to define what a *Qt* widget is. A widget is the atom of the user interface, receiving user interaction events and painting itself in the screen. It is represented by the `QWidget` class and every element of the interface is an example of widget, like the main window, the buttons or the dialogues. Each of these elements are represented by a class which inherits `QWidget`, with a name which starts with a *Q*, followed by a descriptive name, like `QPushButton`, `QLabel` or `QMenuBar`.

We also have to describe how objects communicate in the *Qt* framework. This is done by using the signals and slots mechanism. Signals and slots are a special kind of functions that *Qt* has. A signal is emitted by an object when a certain event occurs and a slot is a function that is called when a certain signal is emitted. The signal which makes a slot be executed is assigned by using a function called `connect`.

Finally, given that *Qt* is a complex library, manually creating a *Makefile* is a difficult task. This is the reason why *Qt* includes a tool called *qmake*, which simplifies the building process. *qmake* uses a *PRO* file, which is very easy to create, to generate the corresponding *Makefile* to build the application.

7.2 Initial implementation

We coded a first version of the viewer with the basic features, that is, being able to load and show up to two 3D models and do basic navigation. In the next lines we have described the stages we needed to create it. The result of this process is shown in the figure 7.1.

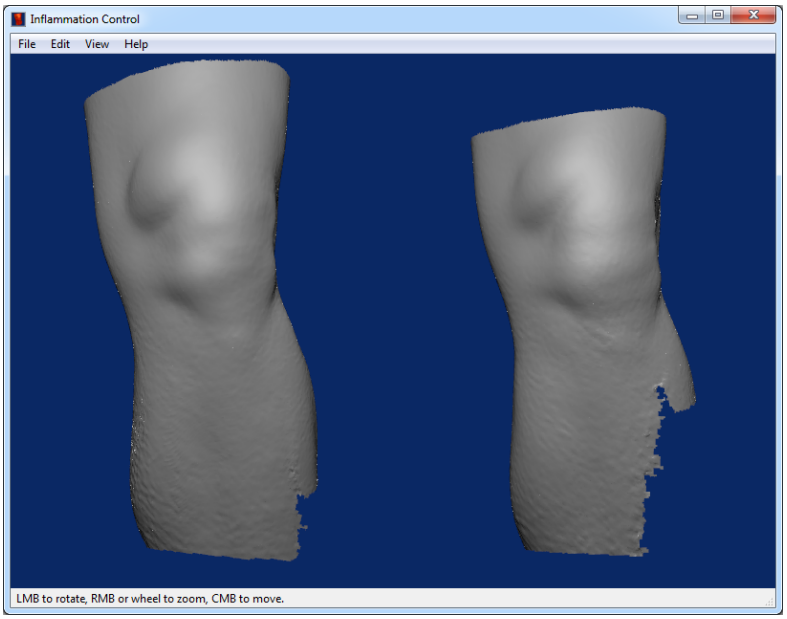

Figure 7.1: Initial implementation of the viewer.

7.2.1 OpenGL window

The most important feature of the viewer is its *OpenGL* window, which is capable of showing up to two 3D models and navigating through them.

Our *OpenGL* window is coded in a class called `ViewerGLWidget`, which inherits the `QGLWidget` class, which is a *Qt* class that creates a widget which is able to render *OpenGL* graphics.

In order to allow it to render what we desire, we have added our own code for the `initializeGL`, `paintEvent` and `resizeGL` functions:

- The `initializeGL` function is called before the first use of `paintGL` or `resizeGL`, and it aims to initialise the *OpenGL* rendering context. In our case, this function sets up the background colour of the window, enables the depth test, loads the shaders that are going to be used and sets up their parameters.

- The `resizeGL` function is called when the widget is resized. In our implementation, we update two variables which store the size of the widget, variables used in the rendering process.

- The `paintEvent` contains the most important part of the *OpenGL*-related code. Its goal is to repaint the widget. Usually, a `QGLWidget` uses the function `paintGL`, its goal being to repaint the *OpenGL* scene. Since it is latex explained that we have extended the viewer to draw 2D graphics in the top of the scene, we have decided to use the `paintEvent` function, since it allows the painting of 2D graphics using the standard *Qt* drawing classes. In our implementation, we set up the viewport first, since its size is not always the same, it depends on the size of the widget and also on whether we show two models or only one. After that, we calculate the model, view and projection matrix, and we pass it to the vertex shader. Finally, we make active the vertex buffer objects of the positions, normals and colours, which are created when the model is loaded. We have to say that in the case that we render two models, the process is executed twice.

Navigation is implemented adding our own code for the `mousePressEvent`, `mouseMoveEvent` and `mouseWheelEvent` functions:

- The `mousePressEvent` function is called when the mouse is pressed. In our implementation, we store the position where the mouse is in the moment of pressing any of its buttons, since we are going to need it to allow navigation.

- The `mouseMoveEvent` function is called when the mouse is moved while a button is pressed. In our implementation, we calculate the vector representing the movement of the mouse since the last time that the function was called. Using this vector,

we update the rotation, zoom or pan depending on whether the button pressed is the left button, the right button or the middle button, respectively. Finally, we store the position where the mouse is when executing the function.

- The `mouseWheelEvent` function is called when the mouse wheel is moved. In our implementation, we check if the movement is positive or negative, and depending on the result, we zoom in or zoom out the scene rendered.

A timer is set and connected to the `update` function so that the widget is repainted as soon as possible. Doing that we assure that the navigation transformations and other operations that may affect the image shown can be seen immediately.

Our models are rendered using a custom vertex shader and a custom fragment shader. The vertex shader is going to calculate both positions and normals of each vertex in the window, applying the model, view and projection matrices, and it passes the data to the fragment shader. The fragment shader calculates the colour of each fragment using the Blinn-Phong model, which has the following formula:

$$K_e + K_a(M_a + I_a) + K_d I_d(N \cdot L) + K_s I_s(N \cdot H)^s$$

Where:

- K_e: material emission.
- K_a, K_d, K_s: material ambient, diffuse and specular.
- s: material shininess.
- M_a: model ambient light.
- I_a, I_d, I_s: light ambient, diffuse and specular.
- N: normal vector.
- L: vector to the light source.
- H: half vector, which is a halfway vector between the viewer and the light source.

Figure 7.2 shows the vectors used. Note that V is the vector to the position of the viewer.

We use this lighting model since it provides specular highlights, which are very useful when it is important to perceive features of the geometry of the models. The material values have been set in a way that specular highlights are visible.

Since most of the changes that the user can to do to the program involve changing

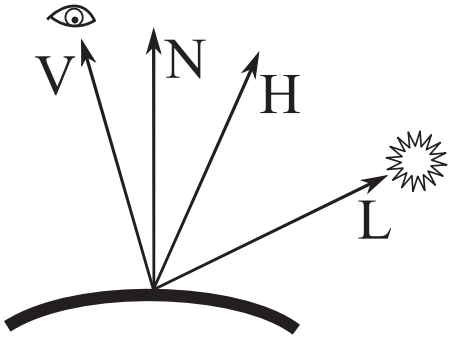

Figure 7.2: Vectors used in the computation of the Blinn-Phong shading model.

something in the *OpenGL* window, we have added functions that execute these changes. These functions have been declared as slots. Examples of slots are the functions that are called when the object shown in the window is changed. Also, some variables that store the state of the window have been needed.

7.2.2 Main window

The main window has been designed using the design function of *Qt Creator*, called *Qt Designer*, which allows to visually create a GUI. The menus, the status bar and the *OpenGL* window have been added in this stage, forming the window that shows up when executing the application. This process creates a *UI* file, which contains all the result of the design of the window.

Code to create the main window has been needed too. The main window is represented by the `MainWindow` class, which inherits the `QMainWindow` class, which is a *Qt* class that creates a main application window.

Menu options for importing models, changing the background colour, resetting the navigation transformations performed or showing the about window were added in this stage, but it was not enough to add them to the user interface, since they have to act on the main window or the *OpenGL* window. It was also needed to create functions that perform the actions that the menu options should execute. Actions that act on the main window, which are the actions that open dialogues, like the about of import model options have its corresponding functions in the `MainWindow` class, connected to the *OpenGL* window, if needed, using slots and signals, while actions that only affect the *OpenGL* window, like the reset navigation transformations option, have its corresponding functions in the `ViewerGLWidget` class.

There were cases where the standard dialogues did not satisfy our needs. In these cases, we had to design our own dialogues. We did this following the same process that we

followed to design the main window, but in these cases we inherited the `QDialog` window. For every kind of dialogue we did then create a new class, which was modified in order to add all the functions that allow setting the initial values, store the values defined by the user or configure the style of the window. We have added a dialogue for the model loading option, and we have added dialogues to the extensions we implement, if needed.

7.2.3 Model representation

We have created a class to handle objects, and have called it `Object`. It contains the position, normal and colour for every vertex, the information about the faces of the model in relation to the vertices, the bounding box and the centroid in order to configure the camera so the whole object is contained in the screen and the variables to store the identifiers of the vertex buffer objects.

In order to show 3D models in the viewer we programmed a way to load them. We have programmed code in order to load models expressed in the *OBJ* format. We have used it because it is extensively used and creating code to parse models expressed in this format is relatively easy.

OBJ loader

The *OBJ* loader is contained in a function which receives as parameter the path of the object that the user wants to load. The function erases the data contained in the instance of the `Object` class, since it could contain another model. Using the *Qt* classes that handle file input, we load the file in a string, parsing it line by line. Since we know that it may have to parse huge files, we have added a progress indicator.

The specification of the *OBJ* format can be found in the document [19]. We have supposed that we are going to use models containing triangular faces only, loading the information about geometry only, this is, vertices and faces. Therefore, we process lines that start with `v` and `f` only, corresponding to a vertex and a face, respectively.

After parsing the file, we calculate the averaged normal of each vertex, that is, the average of the normals to each face which is incident to the vertex, and we give the whole object a grey colour, since colours are used for comparison purposes. Finally, we set up the vertex buffer objects to allow the model to be rendered.

7.3 Extensions

In the previous section we have explained the process of creating a 3D viewer with the basic features needed to solve the main problem of this project, but there are other features that can be added to offer a better solution to the user.

7.3.1 Animations

A very good way to visually show the inflammatory process that occurred between the moments when the two models were captured is to show this process through an animation.

We have done this by using a vertex shader that calculates the position of each vertex depending on a timer. This position is calculated by linear interpolation of the original position and the final position, using the value of the timer as the parameter of the interpolation. The same calculation has been applied to the vertex normal, because it has to be changed since it is involved in the lighting calculations and therefore if affects how the user perceives the geometry of the object.

A dialogue that allows the user to configure the speed of the animation and the kind of cycle that it is going to follow has been created. The `ViewerGLWidget` has been updated, adding a slot to load the shader that performs the animation, a slot to change the properties of the animation depending on the result of the dialogue, and updating the `paintEvent` function in order to allow the shader to receive the final position of the vertices as input.

7.3.2 Overlays

Showing the 3D models in the window could have been enough since we planned to use vertex colouring or animations that allow to visually show the results of the inflammatory process, but it can be important to show additional information. We thought that a good way to show information in the window is to draw it in the *OpenGL* window, over the 3D scene. So, we have made code to draw information in overlays.

Qt provides a class called `QPainter` that allows us to draw everything we need, like lines, shapes or text. We have made extensive use of members of this class to generate the colour scales for the two objects that we want to compare.

The process consisted on modifying the `paintEvent` function of the `ViewerGLWidget`, calling functions to generate rectangles, lines and text giving properties like their position in the widget or their colour in order to draw the colour scale for each of the models. A

slot to allow activating or deactivating overlays has been added too.

Figure 7.3 shows the result of adding an overlay to show the colour scale.

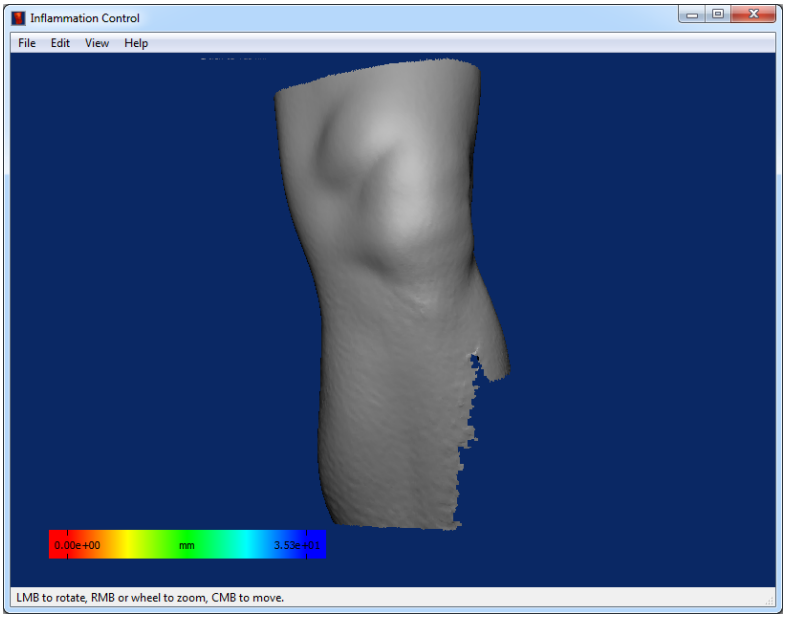

Figure 7.3: Scale overlays shown in the viewer.

7.3.3 Units

In order to do a correct comparison between two different 3D models that represent similar real objects we need them to be scaled properly. Since the units used to express the models could be different, and in fact we will see that they are different depending on the acquisition method, we have to allow the user to select the working unit.

We have done the following process. We decided that the metre would be working unit that we had to use. Therefore, internally everything has been expressed in metres. The user is going to have options to change the unit when loading a model, since he or she has to decide what is the unit used in the file that contains the model. An option to change the unit to express the different comparison measures is provided in the menu, with the corresponding slot in the `ViewerGLWidget` class to change the units shown in scale overlays. The user can choose as units multiples of the metre, that is, apart from the metre itself, decimetres, centimetres and millimetres.

7.3.4 Mesh clipping

The main feature of our program is the ability to compare two 3D models that represent similar objects. There are high chances that one of the differences between the two models are the places where the models have been cut. It would be good to have they cut at the same place, since in the opposite case calculating areas or volumes would not make much sense. That is the reason that led us to implement a way to clip the meshes.

We have added code to clip both models at the same time using a plane. The process consists on deleting the points that are located in one of the subspaces that appear as a result of the plane dividing the space.

A plane in a 3D space can be expressed with an equation of this kind:

$$Ax + By + Cz + D = 0$$

The points that belong to the plane are the points that satisfy the equation.

If we calculate $Ax + By + Cz + D$ with a generic point and it does not belong to the plane, the sign of the result gives information about the subspace created by the plane where the point is. So, the algorithm to clip a mesh using a plane consists on iterating through all the points, checking the subspace where they are and removing the points belonging to one of the subspaces and all faces containing these points. Doing this we do an approximation to exactly clipping the mesh using planes, since we are deleting any face that crosses the plane, but since most of the models that we are going to use are high resolution meshes, the approximation is good enough.

In order to offer the user an easy way to select planes we have made a restriction, which consists on allowing to select planes whose normal is one of the coordinate axis. Two planes are generated, and the user is going to have the ability to move them along the axis that the plane has as normal. One of the planes can move through the positive values of the axis and the other can move through the negative values of the axis.

A dialogue to allow the user to select the mesh clipping planes has been created. It shows buttons to select the axis which is used as normal of the planes and two sliders that determine the position of each of the planes. Slots to show and update the positions of the planes have been created in the `ViewerGLWidget` class. This dialogue does not lock the main window, in order to allow the user to navigate while positioning the planes.

Once accepting the dialogue, the objects are cut using the planes selected. A transformation is needed, since the planes are expressed in the coordinate system resulting of transforming the object to make it fit in the window.

Figure 7.4 shows the result of selecting the option to allow clipping the meshes using

two parallel planes.

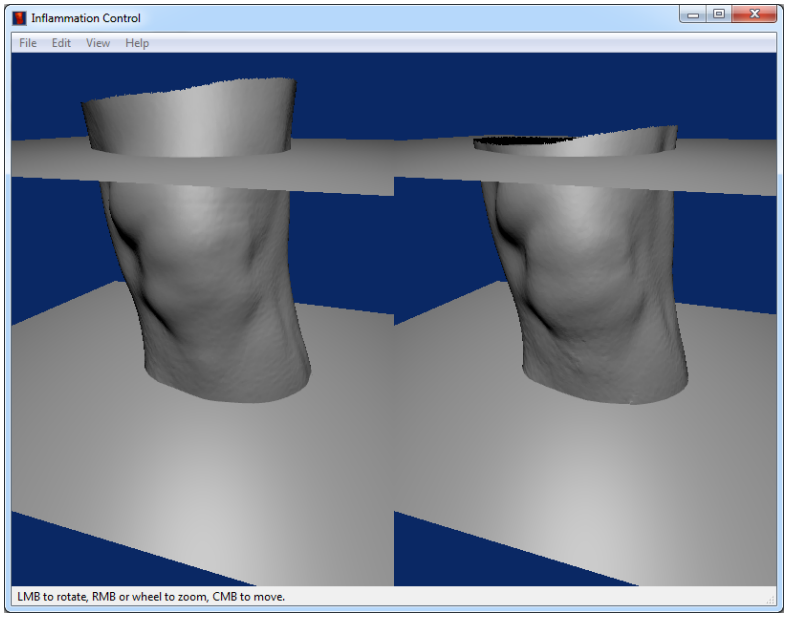

Figure 7.4: Clipping planes shown in the viewer.

7.4 Difficulties

In this section we are going to comment the main difficulties found while programming the first version of the viewer. We think that it is important to describe them since they take an important part in the development of any application. We have had lots of difficulties in this process, there are many that were solved in a few minutes and there are others that took hours to solve. The last kind is what we are going to describe in the following lines.

The first difficulty that we had to face was caused by our lack of experience with *Qt Creator*. Projects created with this program provide two compilation modes, which are *Debug* and *Release*. The *Debug* mode compiles the application in a way that we can use the debugger included with *Qt Creator* to debug it. In our first tests, we compiled using the *Debug* mode since it was the default and everything worked as expected. The problem came when we added time consuming operations like the loading of models. The program much more time than we expected, and trying to improve the code did not help. The cause was the fact that the *Release* mode applies optimisations to the code, while the *Debug* model does not.

Another difficulty was finding how to use some needed *OpenGL* extensions. We fol-

lowed some *OpenGL* programming guides in order to code the viewer, and we had to use functions which were marked as undeclared when compiling. We had used some of these functions in the past, but then we used an external library to use them properly. After some research, we found that *Qt* provides a solution, consisting of using the `QOpenGLFunctions` class, which contains all the functions we need.

Finally, there have been problems when trying to implement overlays. Our first idea, which was inspired by an example provided with the official *Qt* documentation, was to move all the *OpenGL* code to the `paintEvent` function, instead of adding it to the `paintGL` function, which is what we had first. Then, we could add the code that draws the scale, so it would be added over the 3D scene. This way of coding overlays led to constant program crashes or overlays being shown incorrectly, which happened only when they were active. Finally, we realised that the problem could be solved by the unbinding the buffers that had been bound in the *OpenGL* code and making sure that all the vertex attribute arrays had been disabled.

8 Comparison and analysis

In this section we have described the process of exploring algorithms that can be used to align and compare different 3D models that represent the same object and the process of creating code to execute them in our program.

In the *State of the art* section we have mentioned algorithms that we could have used to process the pairs of 3D models. The ICP algorithm to align 3D models and some techniques to compare the models like the calculation of Reeb graphs, medial axis transforms and distance fields, and other proposed measures like the calculation of the area or the volume are examples of methods that we could have used.

In that section we also mentioned two software tools that ease the process of using the techniques presented, which are *MeshLab* and the *Point Cloud Library*. Both tools were used in the development process described in this section.

The first that we had to do was to test the selected algorithms to know if they were useful for our purposes. After that, if the tests were successful, we had to create the code for our program, doing further tests in order to determine the parameters that make the algorithm work best.

We executed the tests with a 3D model of both knees of the same person, obtained with a light-based 3D scanner, using the procedures described in the section *Scanning techniques exploration and implementation*. The model is shown in the figure 8.1. We compared the left and the right knee, mirroring one of them so the shape could be compared properly.

The process described in the previous two paragraphs is documented in the following sections. In each of these sections we have done a detailed description of each of the algorithms and auxiliary data structures used.

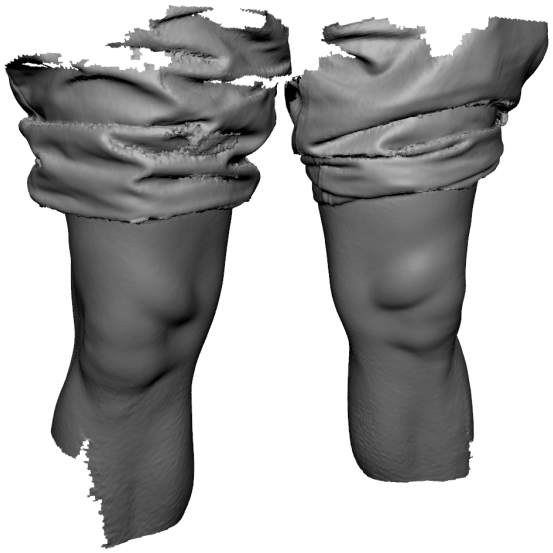

Figure 8.1: 3D model of both knees of the same person, obtained using a light-based 3D scanner, that was used for testing purposes.

8.1 ICP algorithm

We already knew that the ICP algorithm could be used to align two 3D models that represent the same object. Both *MeshLab* and *Point Cloud Library* contain implementations of the ICP, so we did tests with both, but taking into account that *MeshLab* was useful for testing purposes only, since it is not a library that we could call directly from our code.

8.1.1 Description of the ICP algorithm

Given two meshes or point clouds, the steps that the ICP algorithm performs are the following, as described in [8]:

- Select some set of points on one or both meshes.
- Match these points to samples in the other mesh.
- Weight the corresponding pairs appropriately.
- Reject certain pairs based on looking at each pair individually or considering the entire set of points.
- Assign an error metric based on the point pairs.

- Repeat previous steps in order to minimise error metric.

8.1.2 Initial testing

We have performed some tests to check if the ICP algorithm is useful. First, we have used *MeshLab*, since it includes the ICP. Since it could align the models, we created a test program with the *Point Cloud Library* to check how its implementation of the ICP works.

MeshLab tests

MeshLab has the ICP algorithm implemented in the *Align* tool, whose aim is to align two or more models. What we have to do first is to do a rough manual alignment of the models, since the implementation of the ICP algorithm that *MeshLab* contains requires the models to be close enough. This process is shown in figure 8.2, and consists in marking at least four pairs of corresponding points in the two models, in order to calculate a transformation matrix that matches the two sets of marked points and apply it to one of the models.

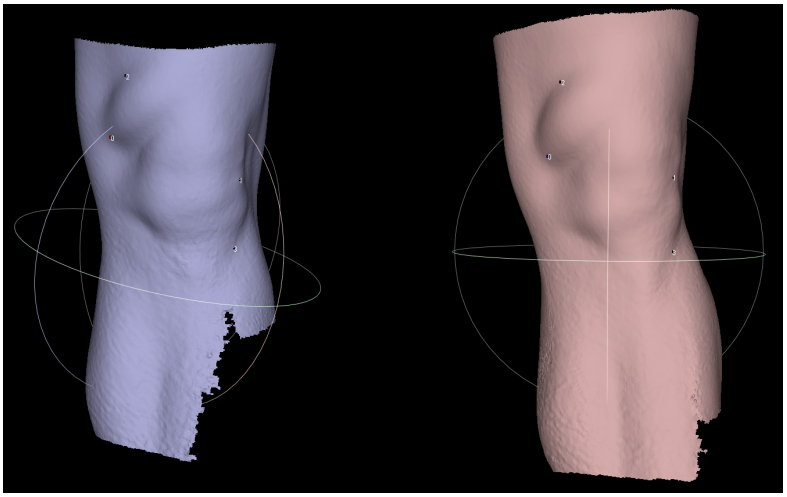

Figure 8.2: *MeshLab* window used to configure the initial alignment for the ICP algorithm. Four pairs of corresponding points are marked.

Once we have the initial alignment, we can execute the ICP, which refines the initial alignment, giving a better one. As shown in the figure 8.3, we can say the alignment is quite accurate.

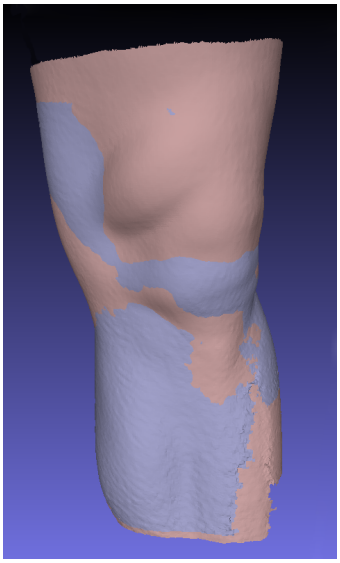

Figure 8.3: Alignment of two knees belonging to the same person done using the ICP algorithm provided by *MeshLab*.

Point Cloud Library tests

Point Cloud Library has its own implementation of the ICP algorithm. In fact, it includes different variants of the ICP algorithm. All of them need two point clouds as input, one being the cloud that is going to be transformed and the other being the target cloud.

The main variant of the ICP is contained in the class `IterativeClosestPoint`, while `GeneralizedIterativeClosestPoint` and `IterativeClosestPointNonLinear` are the two other variants available. We are going to use the first one.

This version of the ICP algorithm has four parameters to allow the user to configure the behaviour of the algorithm, which are the following:

- The maximum distance between two corresponding points, so points whose distance is higher than this value will be ignored in the calculation.
- The maximum number of iterations.
- The value of the difference between the previous and the current transformations that assures convergence.
- The value of the error, calculated as the sum of Euclidean squared errors, that assures convergence.

The last three parameters are closely related to the convergence criteria of the ICP, which are listed in the following lines. The algorithm finishes when any criterion is met.

- The number of iterations has reached the maximum.
- The difference between the previous and the current transformation is smaller than the value the user has imposed.
- The error is smaller than the value the user has imposed.

We have used a ICP example that can be found in the *Point Cloud Library* website, called *Interactive Iterative Closest Point* as base. The program loads two point clouds, that can be obtained through 3D models, and executes an user-defined number of iterations of the ICP. Then, outputs the matrix containing the transformation and shows the result graphically.

In order to test this implementation of the ICP with our models, we just have to run the program, setting our models as input point clouds. It is important to note that in the version of the *Point Cloud Library* that we are using there is support for the *PLY* format only, so we have to convert our models to that format first, which can be done with *MeshLab* or *Blender*.

Figure 8.4 shows the result of running 15 iterations of the ICP using the same models than in the *MeshLab* tests, using the same initial alignment. The alignment looks as good as the alignment obtained using *MeshLab*

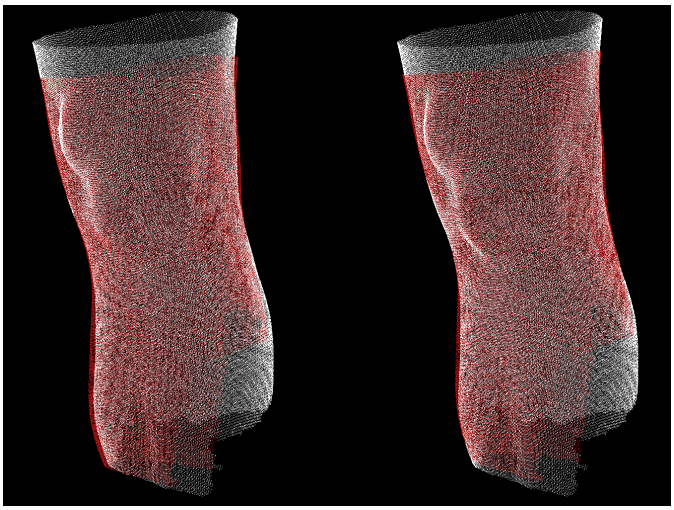

Figure 8.4: Picture in the right shows the alignment of two knees belonging to the same person done using the ICP algorithm provided by the *Point Cloud Library*. Left picture shows the initial alignment.

8.1.3 Tests to determine the parameters

As we have said, the implementation of the ICP algorithm that the *Point Cloud Library* includes can be configured by modifying the values of different parameters. In order to assure the best possible result taking into account the accuracy of the alignment and the time used by the program in the computation we have performed different tests with 3D models of body parts. Doing that we have analysed the impact of the variation of different parameters in the performance of the ICP algorithm.

Initial alignment

In the tests performed when creating the code that contains the ICP in our program we saw that with an initial alignment it was easy to obtain a good alignment. But we still have to determine the importance of the initial alignment in the behaviour of the ICP, that is, we have to check if a good alignment can be obtained if no initial alignment is used. If this happened, we could avoid the user doing a manual alignment before selecting the option that aligns the models.

We have used a model of one of the two knees that have been previously shown. We have cloned it, leaving the first one unchanged and applying a rigid transformation on the second one, that is, a rotation, a translation or a combination of both. Then we have applied the ICP algorithm with different values of the number of iterations and we have evaluated the result. We have set the values of the other parameters in a way that the only convergence criterion that has been applied is the number of iterations.

Before starting with the tests, we have to say that the coordinates system used is the standard one used in graphics, with the X axis going from the origin to the right, the Y axis going from the origin to the top and the Z axis going from the origin to the viewer.

First of all, we have applied a translation to one of the two knees to check its effect in the performance of the algorithm.

The next figures show the results of executing the ICP when translating one of the knees along the X axis, figure 8.5 showing the time spent and figure 8.6 showing the value of the error.

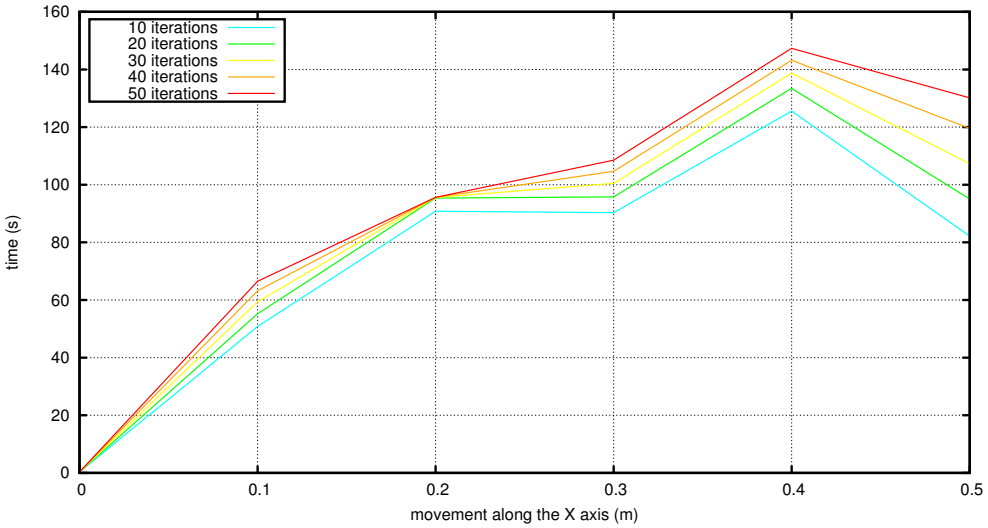

Figure 8.5: Time spent by the ICP when moving one of the knees along the X axis.

This plot shows that the time spent increases with the number of iterations. It also shows that if the magnitude of the movement is higher, the time spent is higher, except for the case where the movement is 0.5 m, which gives less time than in the 0.4 m case.

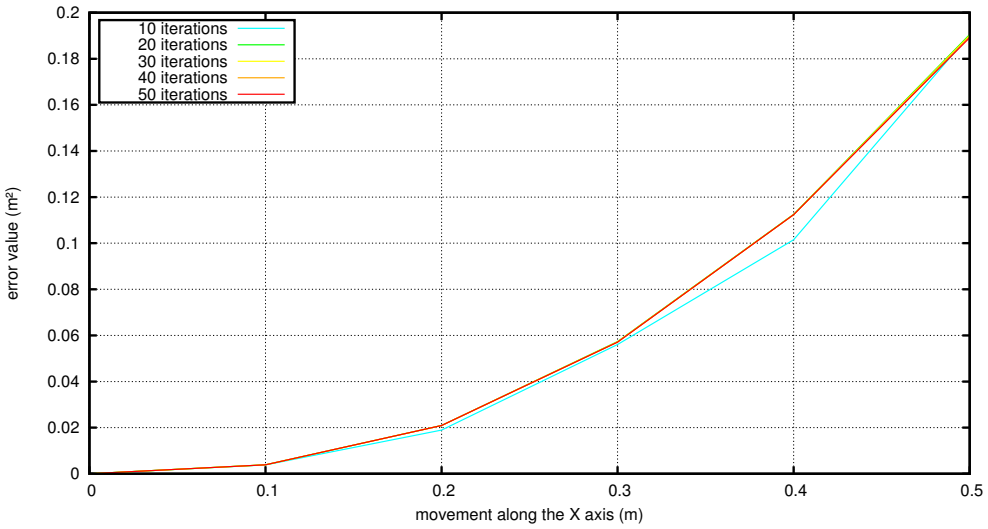

Figure 8.6: Error value obtained when moving one of the knees along the X axis.

This plot shows that the value of the error increases with the magnitude of the movement, but it is almost the same independently of the number of iterations.

Apart from what can be seen in the plots, we have to add a few comments extracted from the visual observation of the results. First of all, increasing the number of iterations

gives better alignments. A value of 10 iterations gives a non acceptable alignment, while 20 and higher values give acceptable alignments, as seen in figure 8.7.

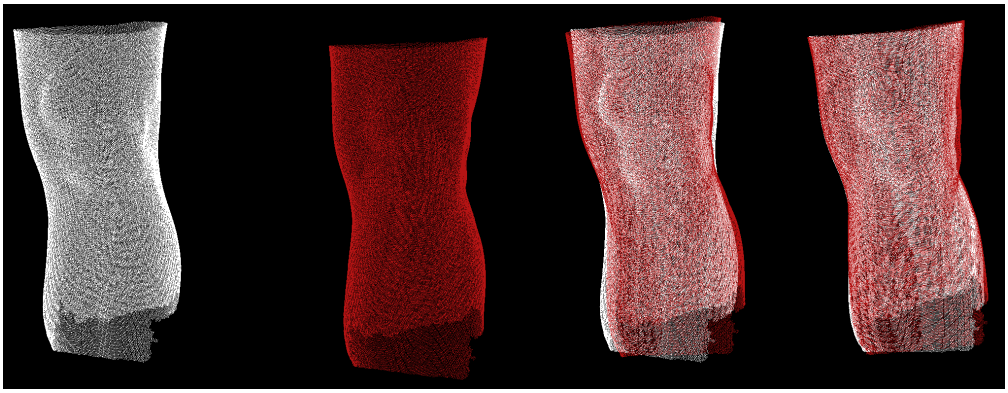

Figure 8.7: The first pair of knees corresponds to a movement of 0.3 m along the X axis. The second pair is the alignment obtained after 10 iterations, while the third shows the alignment after 20 iterations.

The magnitude of the movement affects performance too, since if the magnitude is higher, the results are worse. There is an exception, though. Surprisingly, moving 0.1 m gives worse results than moving 0.2 m, as seen in figure 8.8, and with a high number of iterations we get good alignments for all values of movement except 0.5 m.

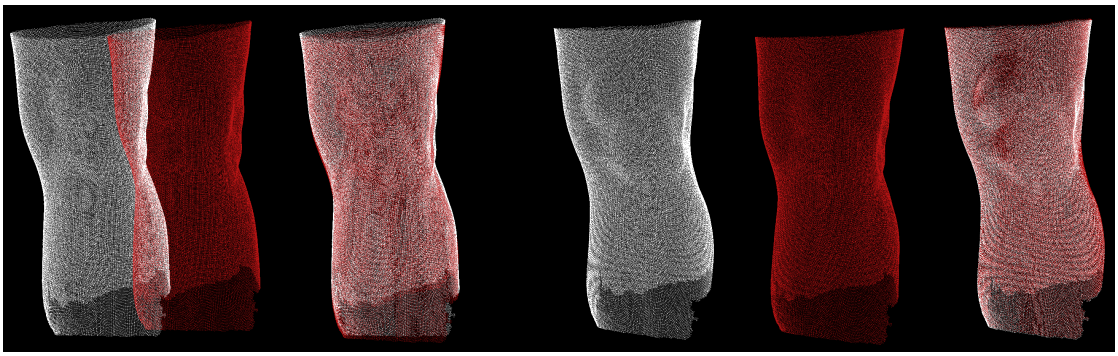

Figure 8.8: The first pair of knees corresponds to a movement of 0.1 m along the X axis. The second pair is the alignment obtained after 20 iterations. The third pair of knees corresponds to a movement of 0.2 m along the X axis. The fourth pair is the alignment obtained after 20 iterations.

The next figures show the results of executing the ICP when translating one of the knees along the Y axis, figure 8.9 showing the time spent and figure 8.10 showing the value of the error.

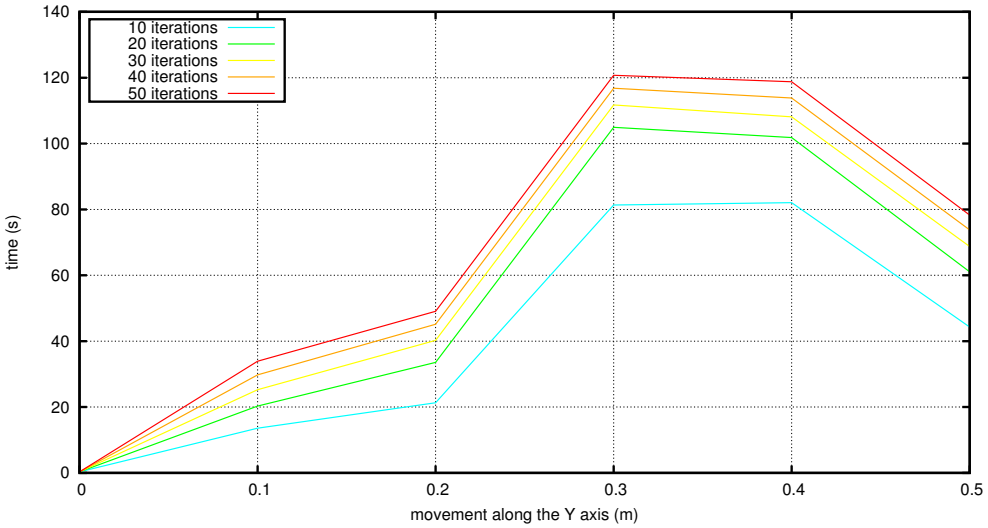

Figure 8.9: Time spent by the ICP when moving one of the knees along the Y axis.

This plot shows that the time spent increases with the number of iterations. It also shows that for the first values of the movement, if its magnitude is higher, the time spent is higher, but for 0.4 and 0.5 it starts to be lower.

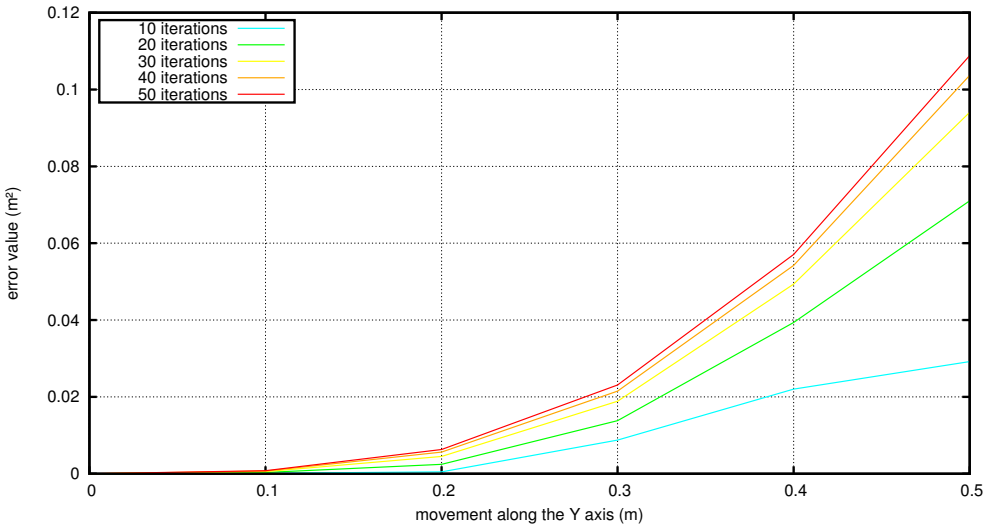

Figure 8.10: Error value obtained when moving one of the knees along the Y axis.

This plot shows that the value of the error increases with the magnitude of the movement, and it also increases with the number of iterations.

Just like in the previous test, increasing the number of iterations gives better alignments. There is a relation between the magnitude of the movement and the quality of the

alignment, since if the magnitude of the movement is higher, the alignment is worst. However, using 50 iterations gives a good alignment for all movement values. So, in this case, it looks like that getting a good alignment is only matter of increasing the number of iterations.

The next figures show the results of executing the ICP when translating one of the knees along the Z axis, figure 8.11 showing the time spent and figure 8.12 showing the value of the error.

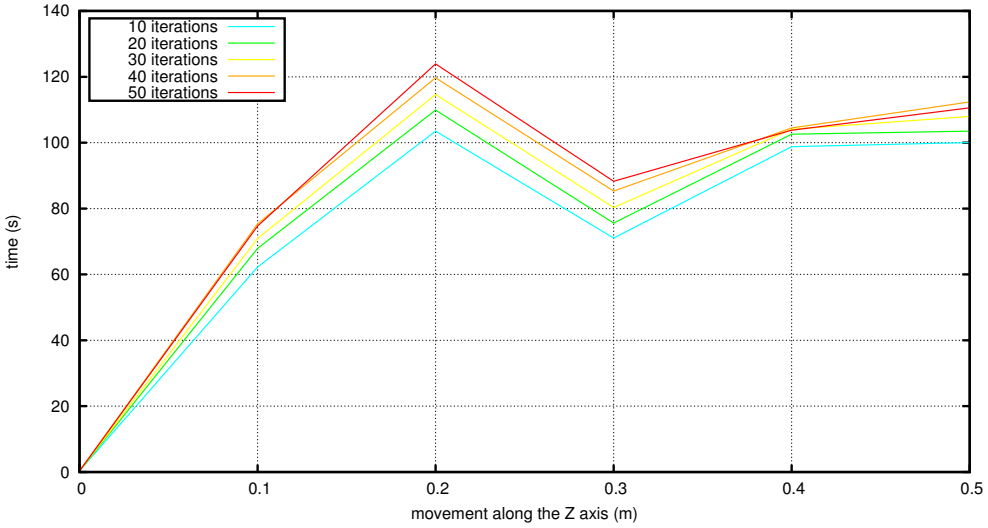

Figure 8.11: Time spent by the ICP when moving one of the knees along the Z axis.

This plot shows that the time spent increases with the number of iterations in most of the cases. For higher values of the movement there is almost no difference between 40 and 50 iterations. It also shows that the time spent increases, then decreases, to increase again and stabilise.

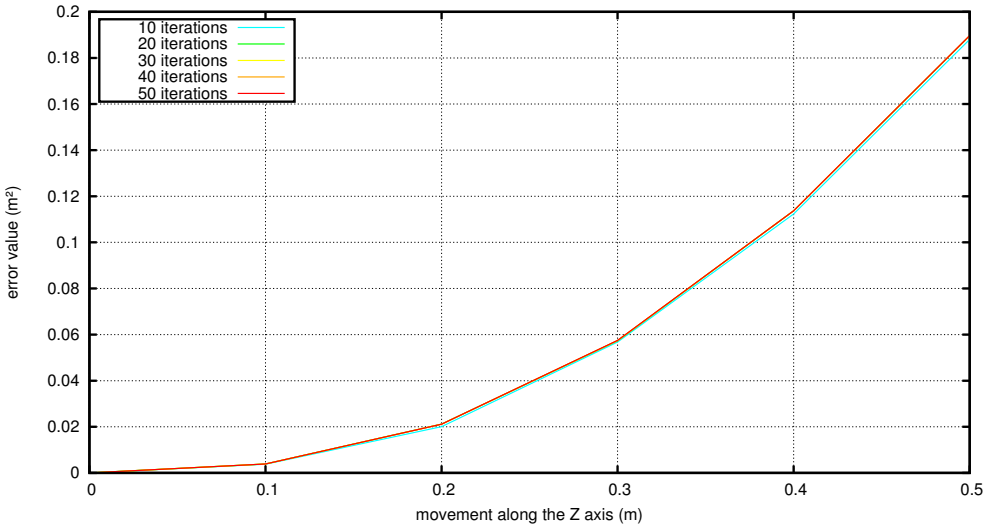

Figure 8.12: Error value obtained when moving one of the knees along the Z axis.

This plot shows that the error value is almost the same for all values of the number of iterations, and it increases with the magnitude of the movement.

Again, we have to add a few comments extracted from the visual observation of the results. First of all, increasing the number of iterations gives better alignments, although in some cases, even setting 10 as the value of the number of iterations gives a good result. Surprisingly, these cases are not the ones with a lower movement magnitude, because, for example, 0.4 m gives better alignments than 0.2 m, as seen in figure.

Given the symmetries of the shape of a knee, we expected the ICP to have a similar behaviour in movements along the X and the Z axis, but we have seen that while the plot of the error value is similar, the plot of time spent is not. We think that this is caused because the shape is roughly symmetrical but not completely symmetrical, therefore moving along a different axis gives different performance.

After that, we have checked how the ICP behaves when the transformation applied is a rotation. In the following tests we have applied a rotation centred in the centroid of the object to one of the two knees.

Figures 8.13 and 8.14 show the result of applying a rotation about the X axis.

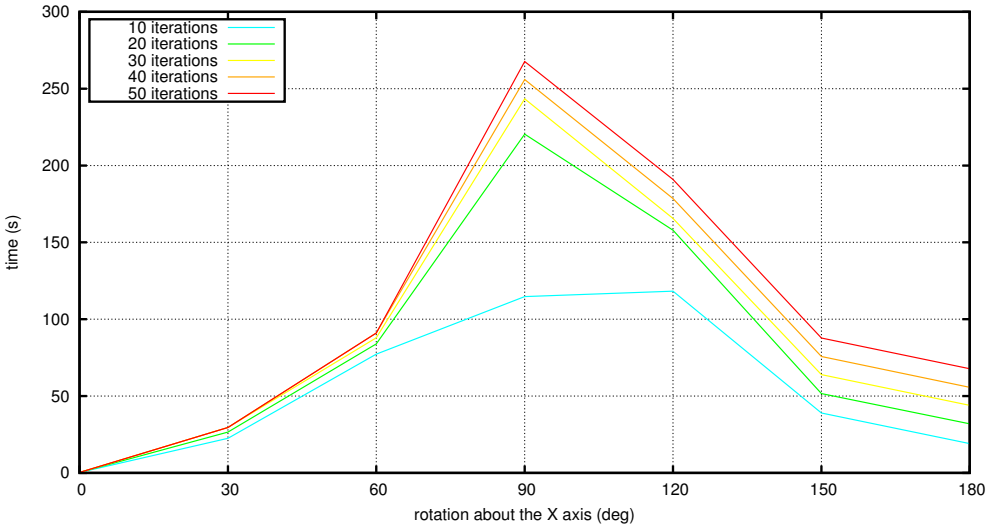

Figure 8.13: Time spent by the ICP when rotating one of the knees about the X axis.

This plot shows that the time spent increases with the number of iterations. it also increases with the degrees of rotation, until the rotation is 90 degrees, since for higher values it starts to be lower.

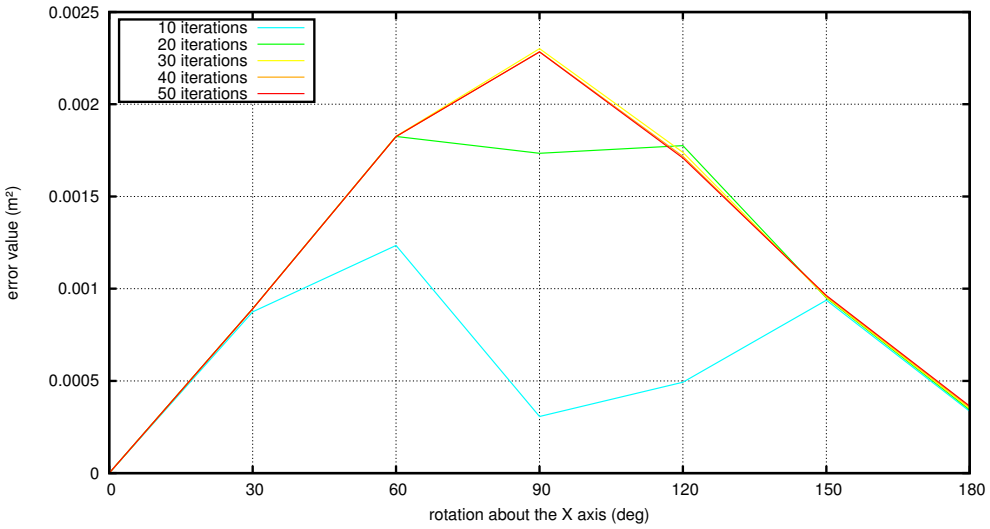

Figure 8.14: Error value obtained when rotating one of the knees about the X axis.

This plot shows that the behaviour of the error value is similar to the behaviour of the time with respect to the value of the rotation, and it has almost the same value for all number of iterations, except for 10, which has a lower value.

Like in the previous tests, we are going to comment other details that are not shown in

the figures. Again, increasing the number of iterations gives better alignments. A value of 10 iterations give a non acceptable alignment, 20 iterations is better but could be much better, and higher values give very good alignments. The degrees that we rotated the model affect the performance too, since values that are closer to 90 make the alignment more difficult. There is also an important detail related with the fact that the shape of a knee is roughly symmetric. A problem arises, since for 90 and higher values the knee is aligned, but it is aligned upside down, as can be seen in figure 8.15. This explains the fact that the maximum time is obtained for 90 degrees.

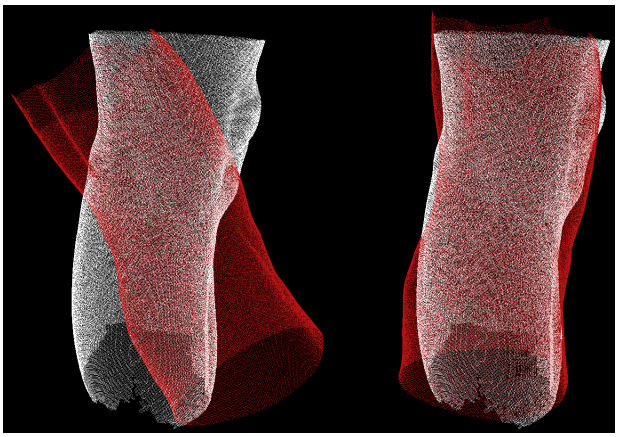

Figure 8.15: In the left, knee in red rotated 150 degrees about the X axis. In the right, the knee after the ICP alignment process.

Figures 8.16 and 8.17 show the result of applying a rotation about the Y axis.

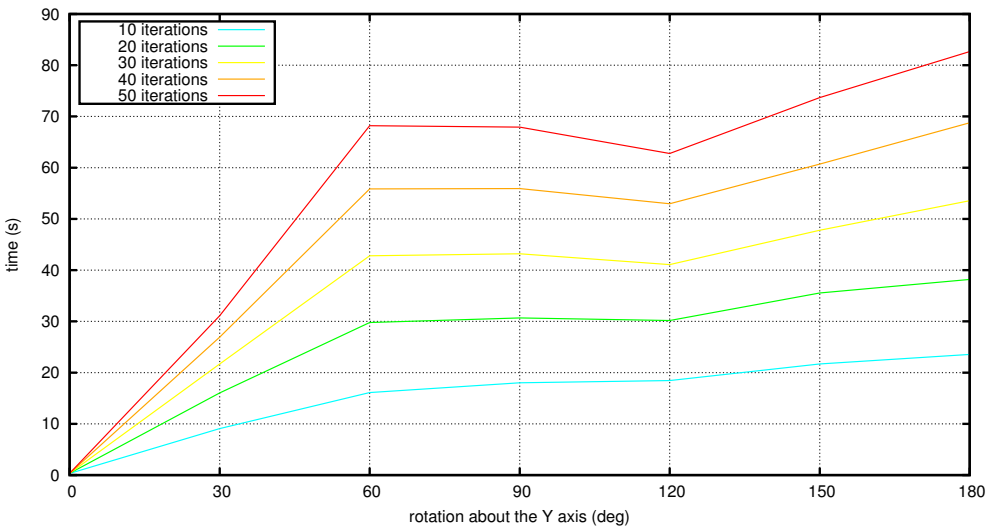

Figure 8.16: Time spent by the ICP when rotating one of the knees about the Y axis.

This plot shows that the time spent increases with the number of iterations. It also increases with the degrees of rotation, except for the interval between 60 and 120, where it seems to be kept, or even decrease.

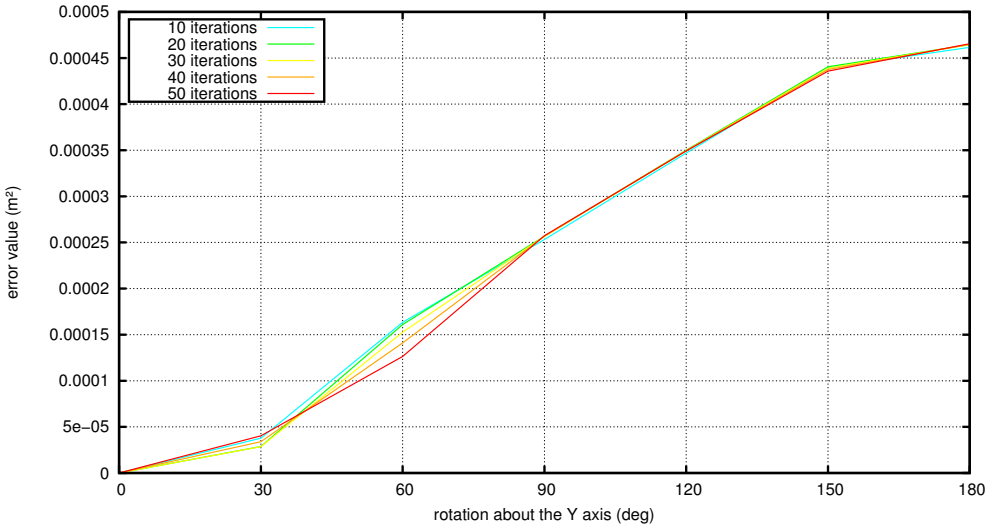

Figure 8.17: Error value obtained when rotating one of the knees about the Y axis.

This plot shows that the value of the error is almost the same for all values of the number of iterations, and it increases with the degrees of rotation.

Apart from the data obtained, we have to comment the visual observations. We have seen that increasing the number of iterations gives better alignments. A value of 10 iterations give a non acceptable alignment, 20 iterations is better but could be much better, and higher values give very good alignments. The degrees that we rotated the model affect the performance too, since values that are closer to 90 make the alignment more difficult. The symmetry of the knee is also a problem. Since the Y axis is not parallel to the axis of the knee, when rotating it, the knee looks bent, and what ICP does is to correct this, as can be seen in figure 8.18, but it does not correct the rotation about the Y axis.

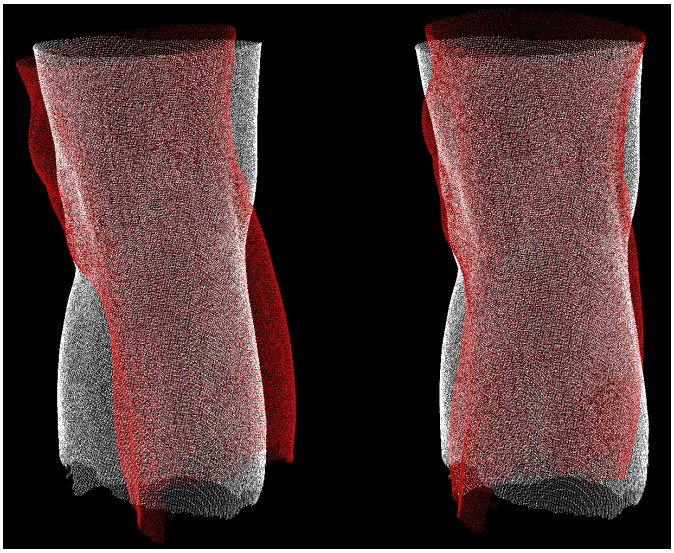

Figure 8.18: In the left, knee in red rotated 90 degrees about the Y axis. In the right, the knee after the ICP alignment process.

Figures 8.19 and 8.20 show the result of applying a rotation about the Z axis.

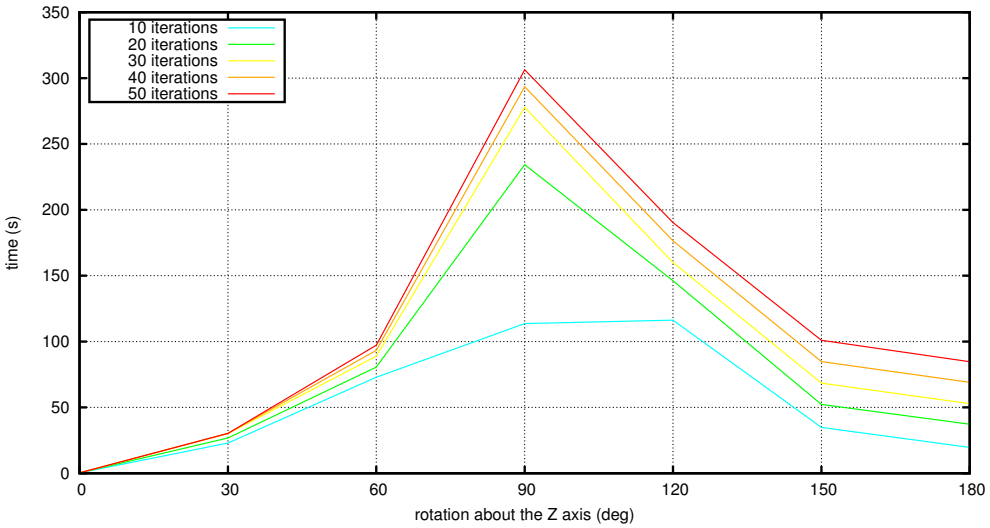

Figure 8.19: Time spent by the ICP when rotating one of the knees about the Z axis.

This plot shows that the time spent increases with the number of iterations. it also increases with the degrees of rotation, until the rotation is 90 degrees, since for higher values it starts to be lower.

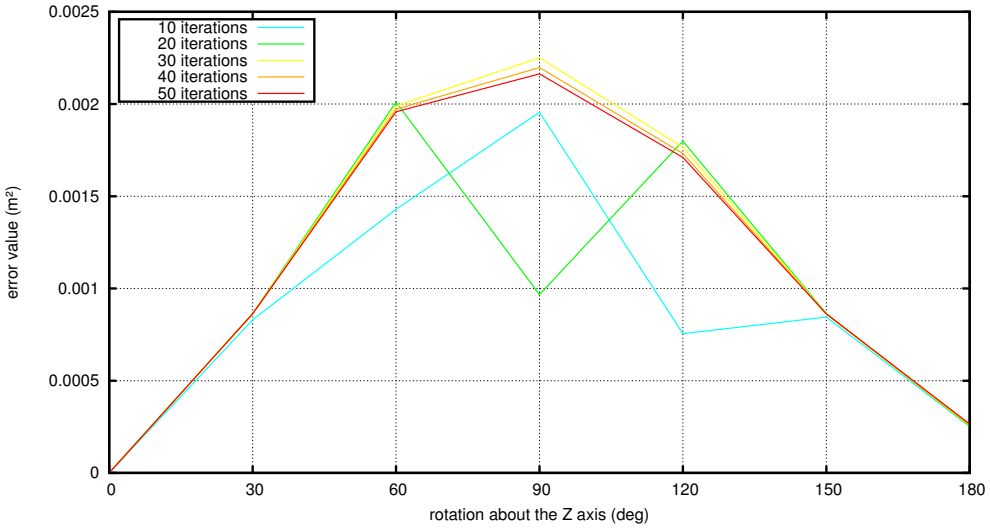

Figure 8.20: Error value obtained when rotating one of the knees about the Z axis.

This plot shows that the behaviour of the error value is similar to the behaviour of the time with respect to the value of the rotation, and it has almost the same value for all number of iterations, except for 10, which has a lower value, and 20 in the 90 degrees case.

The results obtained when rotating about the Z axis are roughly the same than the results obtained when rotating about the X axis, that is, for values of the number of iterations higher than 20 the alignment is good, but for rotations of 90 degrees and higher the result is upside down.

We can conclude that a good initial alignment is the key to get the best possible final alignment, since it does not just save time, but makes it possible for the algorithm to find the right transformation to align the models in a way that we can consider as acceptable, since as we have seen there are transformations that make the alignment an impossible task.

In the next sections we are going to use results obtained in the test described in this section to analyse the impact of the error value and the number of iterations in the quality of the alignment.

Number of iterations

In the tests performed when creating the code that contains the ICP in our program we had to decide a value so we could run the algorithm. We chose a value such that the resulting alignment was good enough to make a comparison, but we did not take

any other decision about this value. In the following lines we are going to compare the results obtained when modifying the maximum number of iterations so that we could decide on using a certain value or give the user freedom to change it.

We can get an idea of how the number of iterations affects the performance of the algorithm by looking at the results of the previous test. When having an acceptable initial alignment, 30 or even 20 iterations should be more than enough. Doing more iterations increases the running time, but there is not a noticeable difference in quality unless the initial transformation magnitude is higher than in the transformations applied in the testing process, fact that can be avoided by determining an initial alignment.

We have not decided on an optimal number of iterations, so we have determined that the best option is to offer the user an option to select the number of iterations. Knowing that when the initial alignment is good the time spent is not as high as in our tests, which have been done with poor initial alignments, it is not a risk to allow the user to define an arbitrary number of iterations. Figure 8.21 shows the time spent by the ICP when aligning two knees that have manually been aligned using *MeshLab*. Even when the number of iterations is high, the running time is not over 20 seconds.

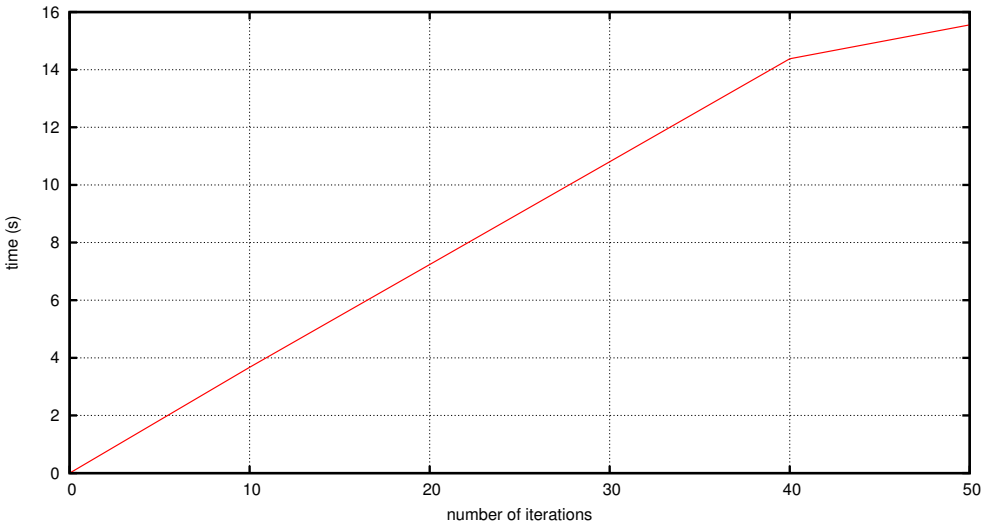

Figure 8.21: Time spent by the ICP when the models have previously been aligned.

Error value

Executing the ICP gives a value as a result, which quantifies the error of the alignment. This value could be used as the value that points out if the alignment is good enough or not.

In order to determine a value of the error that we can consider as being good for our purposes we have to visually check the result of the alignment performed by the ICP algorithm and find its relation with the value of the error.

The tests executed to determine the importance of the initial align can be useful to decide the importance of the error value on the quality of the alignment. We have commented that increasing the number of iterations gives better results. We thought that this fact should be reflected in the value of the error. However, there are cases where the value of the error is almost the same independently of the number of iterations used, as shown in the figures 8.6 and 8.14. Also, we have cases where the alignment is visually better but the error value is worse, for example, we have commented that when moving along the X axis, using 20 iterations gives better results than using 10, but figure 8.6 shows less error for the 10 iterations case, although the difference is small; we have said that when moving along the Y axis increasing the number of iterations gives better results and figure 8.10 shows a higher value of the error for a higher number of iterations.

The value of the error is, according to the *Point Cloud Library* documentation, the sum of squared distances between corresponding vertices from the source model to the target model. That means that if an alignment is better than another one because there are more correspondences it could be that the error was higher because there were more correspondences and therefore, even if every distance was lower, the sum could be higher. This is our interpretation of the cases where a worse alignment has a lower error value.

We can then conclude that the value of the error can not be used to decide whether the alignment is good or not.

8.1.4 Final coding

The code of the ICP that our program uses is contained in a class called `ComparisonFunctions`, which contains all the functions used to compare the two 3D models.

The ICP is executed in a function called `calculateTransformationMatrix`, which has the objects that we want to compare as parameters and the transformation matrix as output.

In this function, we create two point clouds. Then, we fill them with the points of the objects which have been passed as parameters. After that, we set the parameters of the ICP algorithm. Finally, the ICP algorithm is executed and the transformation matrix is set as the return value of the function.

The returned matrix is used to transform the model, being used in a function containing a loop that applies the matrix to all the vertices of the model and calculates the transformed normals.

A dialogue to allow the user to configure the number of iterations has been added.

8.2 Distance field algorithm

We had already explored techniques to align different 3D models, so the next step was to find a way to compare them. Our first approach was the computation of a distance field on each model, using as distance the Hausdorff distance. *MeshLab* has the Hausdorff distance implemented, so has been useful for testing purposes. The *Point Cloud Library* does not have the Hausdorff distance implemented, but it has been easy to create code to calculate it given the different structures that it provides.

8.2.1 Description of the distance field algorithm

In order to compute a distance field, we need to define a distance. We have used the Hausdorff distance. Let A, B be two meshes or point clouds. Then, the Hausdorff distance from A to B is:

$$H(A,B) = \max_{a \in A} \left\{ \min_{b \in B} \{d(a,b)\} \right\}$$

Where $d(a,b)$ is the Euclidean distance between two points in the space.

This gives us the distance between two meshes or point clouds, but it is not useful to compare corresponding zones of two meshes, which is what is needed to evaluate inflammation. What we can do is to measure the Hausdorff distance at each point of one of the two meshes. That is, if $a \in A$, we have to calculate:

$$H(a,B) = \min_{b \in B} \{d(a,b)\}$$

We have to do that for each point of A. That means that we have to use an efficient structure in order to perform a fast search of the nearest point with respect to a point. We are going to use k-d trees. A k-d tree is a data structure designed to organise points in a k-dimensional space. It is a binary search tree, where every node represents a k-dimensional point and every non-leaf node generates a hyperplane that divides the space into two parts, which are represented by the left and right subtrees.

8.2.2 Initial testing

We have performed some tests in order to know if the calculation of the Hausdorff distance is useful to compare two 3D models. We have used *MeshLab* only, since we only want to know if it is useful.

MeshLab tests

MeshLab includes an option to calculate the Hausdorff distance for each point. We have calculated it on the two models that we aligned in the ICP tests performed using *MeshLab*. Once computed, we have to use the results to describe the differences between both models. The easiest way to do it is to paint each vertex with a different colour depending on the distance. The result is shown in the figure 8.22.

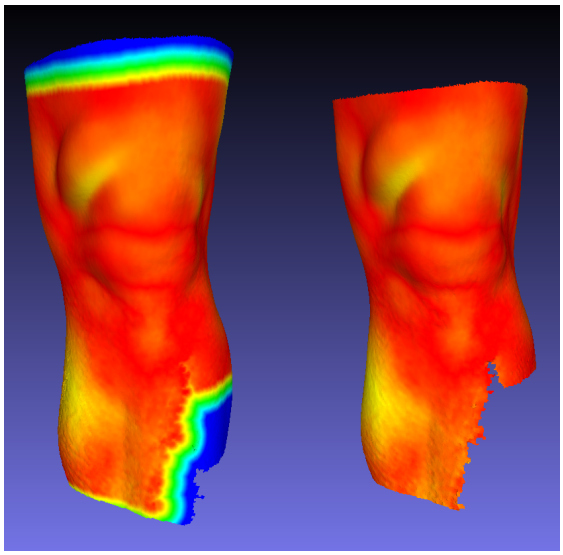

Figure 8.22: Models coloured using the Hausdorff distance calculation provided by *MeshLab*.

8.2.3 Final coding

We have used functions from the *Point Cloud Library* to create the code that calculates the Hausdorff distance in our program. The function, called `calculateHausdorff-Distance`, has been added to the `ComparisonFunctions` class that we created while coding the ICP.

Our function has two parameters, which are the objects that we want to compare.

We create two point clouds and we fill them with the points from the objects just like we did when creating the ICP function.

The *Point Cloud Library* does not include a function to calculate the Hausdorff distance, but it has a function that calculates the nearest k points to a given point in a given point cloud that is represented using a k-d tree and its squared distances to the point. So, we execute this function for each point of both point clouds, and we save the maximum distance found and the point which is at this distance.

After doing that, we colour each vertex depending on the distance. We colour them using the colour scale shown in the figure 8.23, which is the same that *MeshLab* uses, taking 0 as the minimum and the maximum distance obtained as the maximum of the scale.

Figure 8.23: Colour scale used by *MeshLab* and by our program to show the distances.

Figure 8.24 shows the result of computing the distance on the same models than we used while doing tests with *MeshLab*. We can see that the result is not exactly the same, this happens because we are not using the maximum and minimum values of the colour scale in both cases.

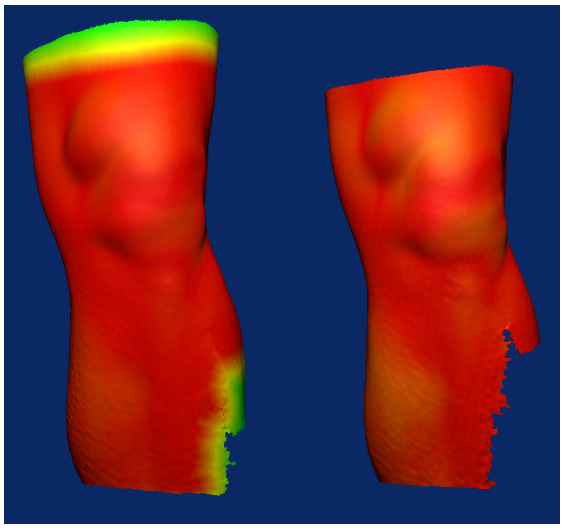

Figure 8.24: Models coloured using the Hausdorff distance calculation coded using the *Point Cloud Library*.

A menu option has been created in order to allow the user to change limits of the colour

scale of both models through the creation of a dialogue.

A last change has been introduced. Since red is usually classified as a warm colour and blue is often classified as a cool colour, we are going to flip the scale, so the zone that seems to have higher colour temperature is the zone suffering most of the effects of inflammation. Figure 8.25 shows the results of using the new scale.

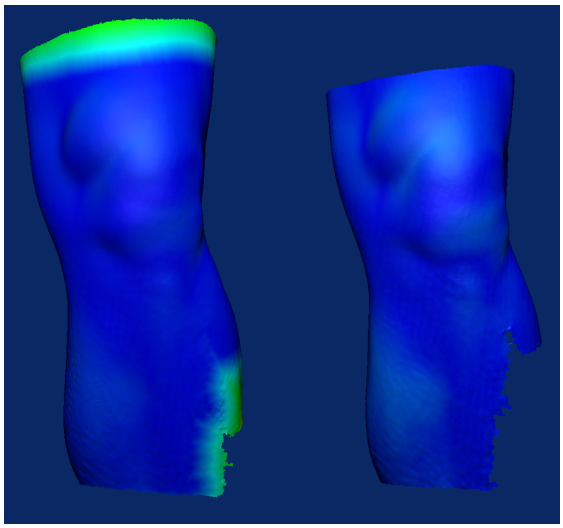

Figure 8.25: Models coloured using the Hausdorff distance calculation coded using the *Point Cloud Library*, but using a flipped scale.

8.3 Computing additional measures

In order to do a more complex analysis of the differences between two models, we decided to calculate additional measures apart from the Hausdorff distance.

We proposed measuring the area and the volume of each model in order to calculate the difference between the values obtained. While we could not tell if these measures were going to help to evaluate the inflammation level we thought that it was worth adding them since additional information is always welcome and it was easy to code them. We have to remark that these two measures need to have the clipping of meshes using planes implemented since it has no sense to compare the area or the volume of two models that have not been clipped using the same planes.

8.3.1 Area calculation

Calculating the area of the surface represented by a 3D model is an easy operation. A 3D model M is composed by a set of faces. So, its area is:

$$A(M) = \sum_{F \text{ face of } M} A(F)$$

In our case, the faces are triangular faces. The area of a triangle in a 3D space can be calculated using the cross product. If a, b and c are the vertices of a triangle T, its area is:

$$A(T) = \frac{1}{2} \left| \vec{ab} \times \vec{ac} \right|$$

And this is easy to code since for each model we store information about the vertices and the faces.

8.3.2 Volume calculation

As it could be expected, the calculation of the volume needs a closed surface. In most of the cases, our surface is not going to be closed, so we are going to use the planes used to clip the mesh as the caps of the model. We already said that the clipping operation was not an exact clipping but an approximation. While this may affect the value of the volume, the difference is going to be negligible.

We have used the technique from [20] to calculate the volume. This document presents a method to calculate physical properties of 3D meshes like the mass, the inertia tensor and the centre of mass. One of these values, the mass, can be used to calculate the volume, since for a solid of density 1, the mass and the volume have the same value, apart from the change of units.

The mass, and subsequently, the volume, can be calculated using the following integral:

$$\int_V 1 dV$$

Where V is the region bounded by our mesh.

Using the divergence theorem, the volume integral can be converted to a surface integral, and we have:

$$\int_V 1dV = \int_V \nabla \cdot F dV = \int_S N \cdot F dS$$

Where S is the boundary of the volume V, F is a function such that $\nabla \cdot F = 1$ and N is a function that returns an outward pointing unitary surface normal.

Since in our case the boundary is composed by triangular faces, we have:

$$\int_S N \cdot F dS = \sum_{T \text{ face of } S} \int_T N_T \cdot F dS$$

Where N_T is the outward pointing unitary face normal of T.

We propose using the following functions as F, $(x, 0, 0)$, $(0, y, 0)$ and $(0, 0, z)$, which have divergence equal to 1.

In the first case, we can write the last expression we presented as:

$$\sum_{T \text{ face of } S} \int_T N_T \cdot (x, 0, 0) dS = \sum_{T \text{ face of } S} N_T \cdot (1, 0, 0) \int_T x dS$$

Analogously, it can be calculated for the other two functions.

After a few calculations that are described in the document, we obtain the formula to calculate the integral of each triangular face:

$$N_T \cdot (1, 0, 0) \int_T x dS = \frac{n_x}{6} \sum_{v \text{ vertex of } T} v_x$$

Where n_x is the value of the X coordinate of the normal obtained by calculating the cross product of two edges of the triangular face and v_x is the value of the X coordinate of v.

Therefore the volume can be calculated as:

$$\sum_{T \text{ face of } S} \frac{n_x}{6} \sum_{v \text{ vertex of } T} v_x$$

Since the faces that form the cap of the object after clipping it using planes are not created, we have to find a way to calculate the volume without using them. Its normal

is the normal of the clipping planes, and it is parallel to one of the coordinate axis, so we can select the appropriate function for F so that the value of the calculation for these faces is equal to 0.

8.4 Difficulties

Just like in the *Viewer design and implementation* section, difficulties take an important part in the development. We have described them in this section.

There has not been any difficulty regarding the *MeshLab* tests, just as expected, since it is a stable program and it is not difficult to use. The most important difficulties appeared when we started creating code using the *Point Cloud Library*.

Regarding the programming process, there has been a remarkable difficulty only, but we needed to solve it in order to use the *Point Cloud Library* in our project. It appeared when we tried to use code with functions and data structures of the *Point Cloud Library* in our application. Code examples of the library are designed to be compiled with *CMake*, which is a program that allows managing the process of building software, but since we are developing a *Qt* application, we are using *qmake*. We had two options, which consisted in either creating a *CMake* file to compile our program, which would involve a learning process to understand how *CMake* works or finding a way to modify the *PRO* file, which is the source file that *qmake* uses, so that all dependencies could be found. We decided on doing the second option, since it looked easier. It was not an easy process, since we had to figure what libraries from the *Point Cloud Library* we were using and had to be included, but we managed to do it right it and compile the application with the new code we had added.

Finally, we have to comment a strange problem that affects the ICP algorithm that the *Point Cloud Library* includes. In the first stages of testing, we had the scanned model that we have shown, and the units it used were millimetres. Unless we had to compare models expressed in different units, there should not be any problem with using any unit, or at least this is what we intuitively thought. Surprisingly, the ICP does not work properly if you use a model expressed in millimetres, that is, it does not seem to do anything even if you run it with a huge number of iterations. If the same model is resized so that is expressed in metres, doing only one iteration gives a result which is better that what you obtain when running several iterations of the ICP with the model expressed in millimetres. In our program we convert everything to metres so that we can load models expressed in different units so we do not have this problem, but we thought that it had to be mentioned.

9 Scanning techniques

We have described the process of creating a program that is able to compare different 3D models in order to compare them and show the user key information so he or she can evaluate an inflammatory process. However, we can assure that this application is worthless if we do not have techniques to obtain 3D models from real body parts. This section aims to describe the techniques used to obtain 3D models so that they can be used in our program.

9.1 Light-based 3D scanner

Among all the 3D scanning techniques that we have access to, the use of a light-based 3D scanner is the most precise one. In the following lines the scanning process is described.

We have used the scanner from the research laboratory of the MOVING research group, called Centre de Realitat Virtual. It is a light-based 3D scanner from *Artec 3D Scanners*, called *Artec MHT*. An advantage of this scanner is the fact that it can texture the model obtained, but since in this project we only need geometry, we are no going to use the texturing feature.

The scanning process is simple. First, we have to open the scanning software, a program provided by *Artec 3D Scanners* that shows in real time the result of the scanning process and some other relevant data. After that, we select the option to start the scanning process. This option makes the scanner start its working process, which consists on emitting a light pattern, a checked pattern, every few milliseconds. Using the result of projecting this pattern on the object the scanner is able to retrieve the geometry of the object. The scanner has a button, once pressed, the program starts to save the scanning result. With the button pressed, we have to move the scanner around the object, making sure that the whole object has been captured. Once finished, we have to release the scanner button and stop the scanning process in the program. The data obtained has a very high amount of noise, and the different captures are merged using an algorithm that makes a fast processing to show the result in real time, so the program provides a few options to improve the quality of the resulting model. The different captures are aligned again, but this time using a better algorithm. The triangular mesh is optimised by applying processes that, for example, fill holes. Finally, the model is ready to export. It can be exported to several formats, including *OBJ*. The coordinates

of the model are expressed in millimetres.

We used this technique with three different people, thus obtaining three pairs of knees. They are shown in the figure 9.1.

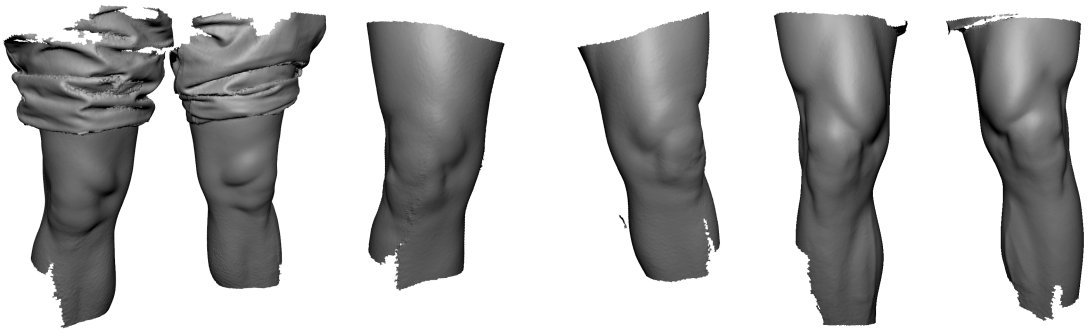

Figure 9.1: Three pairs of knees obtained using the light-based 3D scanner.

In order to graphically show the accuracy of this method we have figure 9.2, which shows the comparison between a scanned knee and its real counterpart.

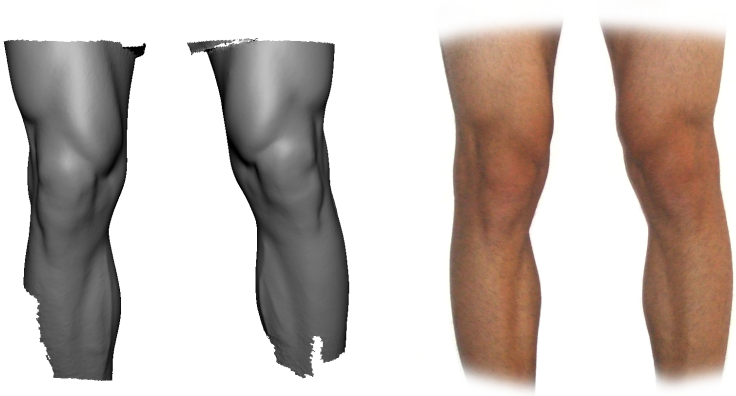

Figure 9.2: A scanned pair of knees and its real counterpart. Apart from evident differences caused by colour and lighting they are nearly identical

9.2 Kinect scanning

The light-based based 3D scanner can generate very precise models, but it has a problem, which is its high price, so it is not a device that is going to be available for everybody. An alternative is the use of a *Kinect*. While it is not as precise as the light-based 3D scanner, it has two advantages, it is affordable for most of the people and most of the people owning a *Xbox* own it too.

We have mentioned that there are two versions of the *Kinect*, the *Xbox 360* version and the *Xbox One* version, the latter having better specifications, like higher precision of the depth camera. Unfortunately, we have not had access to the last version, so we had to use the *Kinect* for *Xbox 360*, which can be used in a *Windows* platform using an older version of the *Kinect SDK*.

Figure 9.3 shows the version of the *Kinect* that have used.

Figure 9.3: *Kinect* for *Xbox 360* device.

Kinect SDK includes plenty of tools and code examples to use the different features that a *Kinect* device offers. We are going to use *Kinect Fusion*, a tool that is capable of generating a 3D model using the data obtained from the depth camera of the *Kinect*, which is shown in figure 9.4

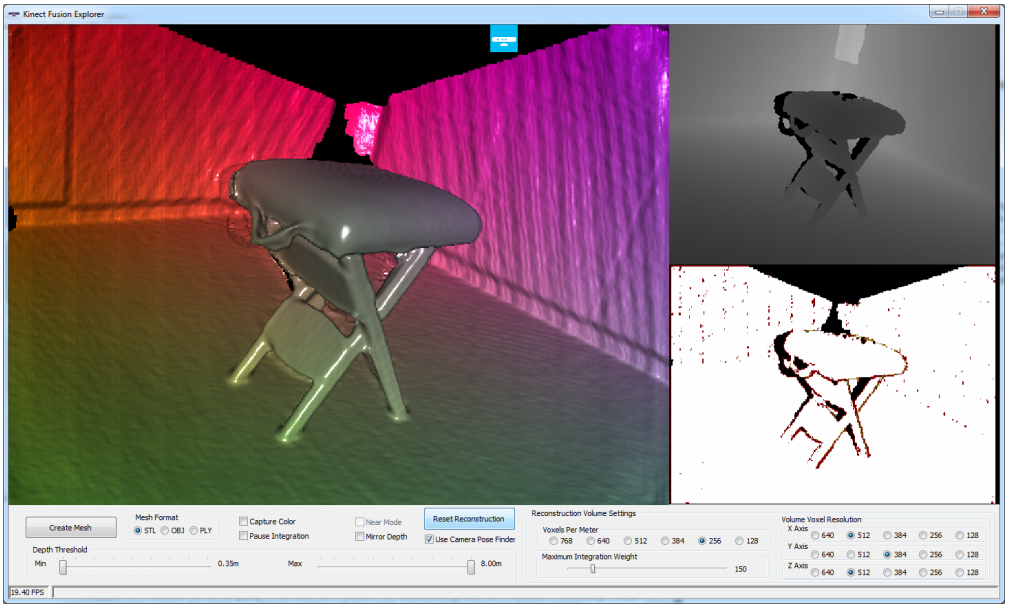

Figure 9.4: *Kinect Fusion* in action, scanning a chair.

The scanning process is similar to the process that we had to follow when using the light-based 3D scanner. We just have the move the *Kinect* around the object that we

want to scan, ensuring that all areas of the object are covered. While doing that, the 3D model generated is shown in the main window of the program. Obtaining a good scan is a tricky process, since it is easy for the program to lose track of the zone that it is scanning in relation to the data already obtained. After finishing the scanning process, it is time to export the model. It can be exported to several formats, including *OBJ*. The models obtained have coordinates expressed in metres.

Figure 9.5 shows a 3D model generated using this technique, compared with the model obtained with the light-based 3D scanner. Although the quality of the model is much lower than the quality of models obtained with a light-based 3D scanner, but taking into account the difference of prices between both devices and the fact that we are not using the latest version of the *Kinect*, we can conclude that the result is very good.

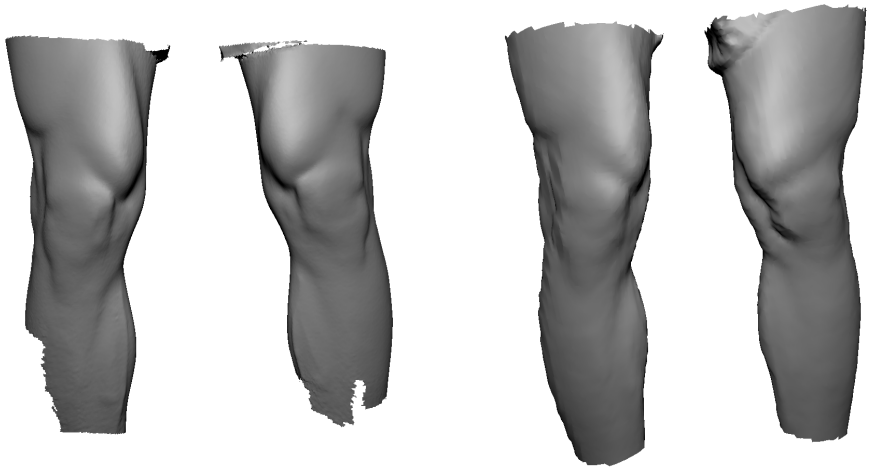

Figure 9.5: In the left, knees scanned using the light-based 3D scanner. In the right, knees scanned using *Kinect Fusion*.

However, these models are not useful for our purposes, since swelling caused by inflammation usually causes differences of a few millimetres and we can see that the precision of the models generated makes detecting these difference an impossible task. Also, it is very difficult to obtain 3D models with *Kinect* with a minimal amount of noise, which makes things even more difficult.

9.3 Multi-view stereo

We would like that our application could be used by everybody having access to a computer, and in fact we are in this case, but there is a problem, since not everybody has access to a 3D scanning device. We know that a light-based 3D scanner is not affordable in most of the cases, a *Kinect* is, but only *Xbox* users are going to own it.

However, most of the people have access to a photograph camera or any other device that can be used to take photographs, so a technique that could reconstruct a 3D model from photographs of the real object would be very useful. This technique is called multi-view stereo.

As we have said, *Autodesk* provide an application, called *123D Catch* that does exactly this. In order to create the 3D model we just have the open the program and load the images taken. They are then uploaded, so that they can be processed in the cloud to generate a 3D model from the pictures sent. This process takes a few minutes and after that, the 3D model is available for download.

Using more than 30 photographs, we obtained the following result, shown in figure 9.6. As it can be seen, not only the legs are reconstructed, but all the surrounding objects, each one of them being textured. Also, we can see that the points where the photographs were taken have also been positioned in the scene. We can observe that it is far from being perfect, but given that it has been one of our first tries on capturing without any experience, we can say that the result is very good.

Figure 9.6: Legs and surrounding objects reconstructed by *123D Catch*.

We can even obtain a more refined version of the mesh changing its quality and selecting the maximum value. However, even doing that, the precision is worse than the obtained with *Kinect*, and it has an added problem, unless an object is added to the scene to calibrate the scale, the scale is arbitrary, meaning that the measures obtained in the program are going to be wrong. Figure 9.7 shows the resulting mesh after removing all the unnecessary parts.

Figure 9.7: Model obtained with *123D Catch* after the removing the parts which are not needed and selecting the maximum quality for the mesh.

Although the mesh resolution is very high, there is a lack of precision which is clearly seen, and it becomes more evident when the object is not textured.

10 Tests and results

In this section we have performed tests that are going to tell if the program is capable of solving the problem that led to its creation. We have used pairs of 3D models obtained from real body parts and we have described the information that has been obtained from the measures that the program computes. To assure the best possible result, we have done a manual alignment by matching points in *MeshLab*.

For the first tests, we have used the pairs of knees obtained using the light-based 3D scanner, shown in the figure 10.1, in left to right order.

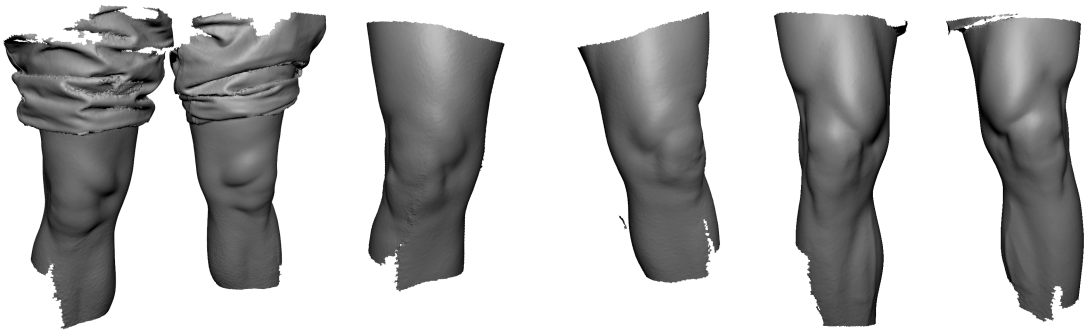

Figure 10.1: Three pairs of knees obtained using the light-based 3D scanner.

After that, we have done a test with the models obtained with *Kinect* and multi-view stereo.

The procedure that we have followed in all tests is the next:

- Load the 3D model of both knees in *MeshLab*, divide it in two models, each of them containing a knee, mirror one of them and make a manual alignment.
- Load the two models created in our application.
- Clip them using planes to remove the zones that are not needed.
- Align both models, executing 25 iterations of the ICP.
- Clip them again using planes, since the alignment process transforms one of the models and we need them to be cut by planes with one of the coordinate axis as

a normal to assure that the volume calculation is correct.

- Calculate distances, areas and volumes.

Before starting with the tests, we have to remark the fact that the right knee of the second model is very noisy and has holes. We know that this has led to incorrect results, especially in the calculation of volumes and areas, but we want to show what happens when the model is not perfect.

10.1 Left and right part of the same person

In this test we have used our program to compare the left and the right part of the same person, having mirrored one of them first. Doing that we are going to be able to tell if there are significant differences between them to decide if one of them is suffering from inflammation.

10.1.1 First pair

After computing the distances, we obtain the following results, shown in figure 10.2.

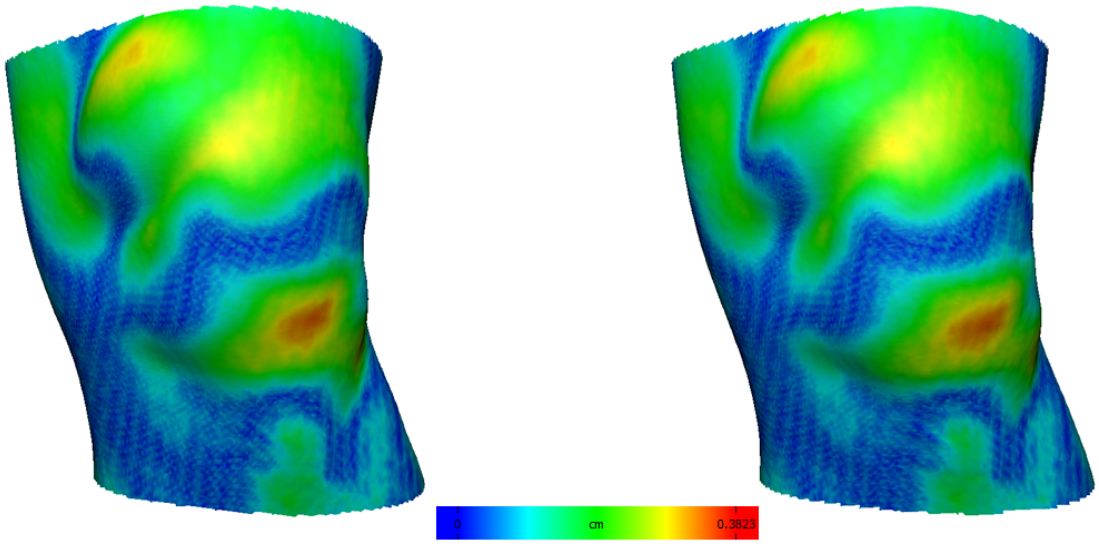

Figure 10.2: Computation of the distance for the first pair of knees.

The area of the left knee is 624.2 cm^2, while the area of the right knee is 624.5 cm^2; the difference is 0.3 cm^2.

The volume of the left knee is 1867 cm^3, while the volume of the right knee is 1864 cm^3; the difference is 3 cm^3.

The animation shows that clearly most of the differences spotted are caused by a slightly different position of the knee, since there is not only a zone of the knee affected, but the whole knee.

10.1.2 Second pair

After computing the distances, we obtain the following results, shown in figure 10.3.

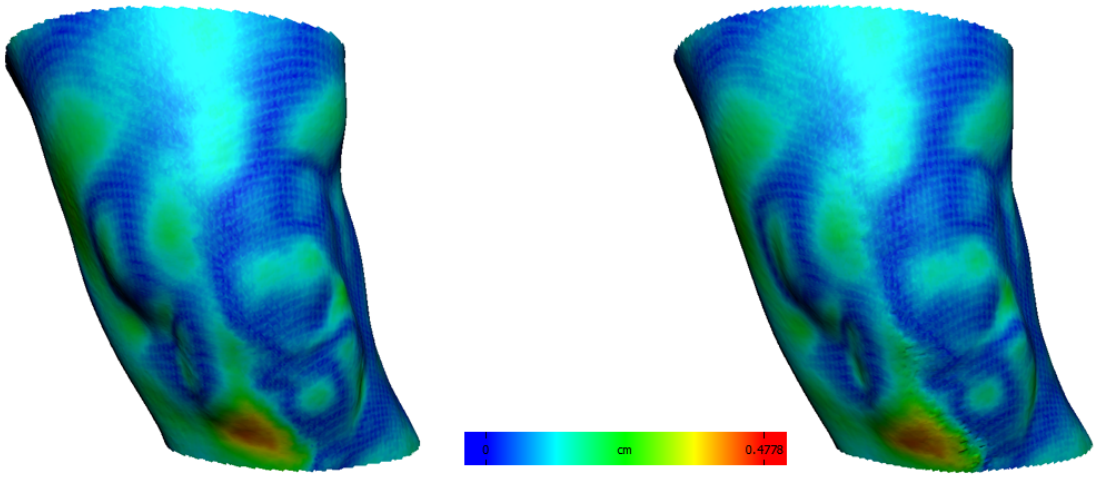

Figure 10.3: Computation of the distance for the second pair of knees.

The area of the left knee is 789.9 cm^2, while the area of the right knee is 873 cm^2; the difference is 83.1 cm^2.

The volume of the left knee is 2465 cm^3, while the volume of the right knee is 2510 cm^3; the difference is 45 cm^3.

The animation shows that changes between both knees are very subtle, and they are not focused in a zone, since they cover most of the knee. Apart from that, there are huge changes in the noisy zone.

10.1.3 Third pair

After computing the distances, we obtain the following results, shown in figure 10.4.

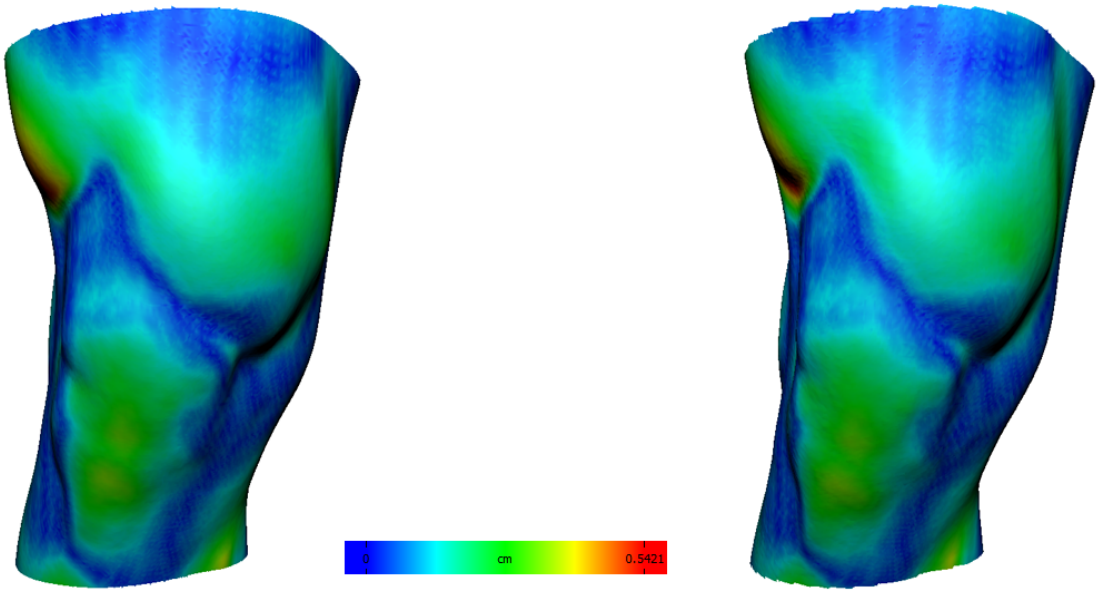

Figure 10.4: Computation of the distance for the third pair of knees.

The area of the left knee is 753.7 cm^2, while the area of the right knee is 753.2 cm^2; the difference is 0.5 cm^2.

The volume of the left knee is 2045 cm^3, while the volume of the right knee is 2014 cm^3; the difference is 31 cm^3.

In the animation we can see that the zone with a bigger distance seems to be noticeably different in both knees, but since the whole knee changes from a knee to the other one, we can say that it is caused by a different position of the knees.

10.2 Same part with a simulated deformation

In this test we have evaluated the results that our program gives when comparing a body part and a copy of it with a deformation created by a 3D modelling program, *Blender* in this case, in order to simulate inflammation.

The procedure in this case involves loading one of the knees in *Blender*, which is the left one, picking a vertex and moving it a few millimetres, using the *Proportional editing* mode, which causes the surrounding vertices to be moved too, giving the appearance of suffering from an exaggerated inflammation.

Figure 10.5 shows the result of applying deformations to the models.

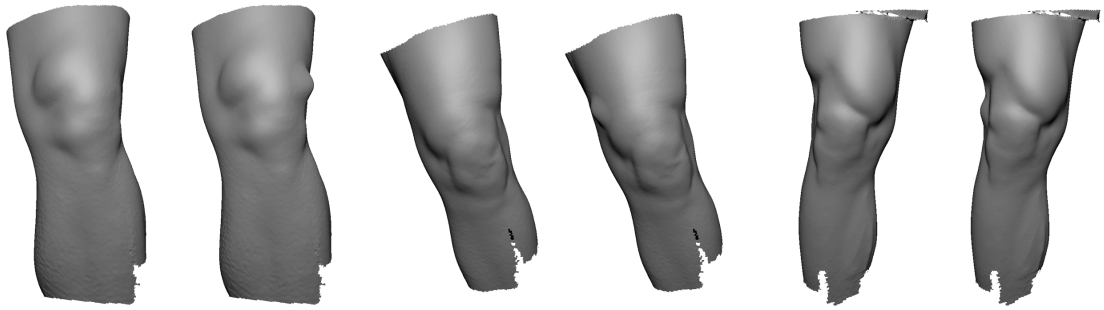

Figure 10.5: Three pairs of knees; non deformed, in the left, and deformed, in the right.

10.2.1 First pair

After computing the distances, we obtain the following results, shown in figure 10.6.

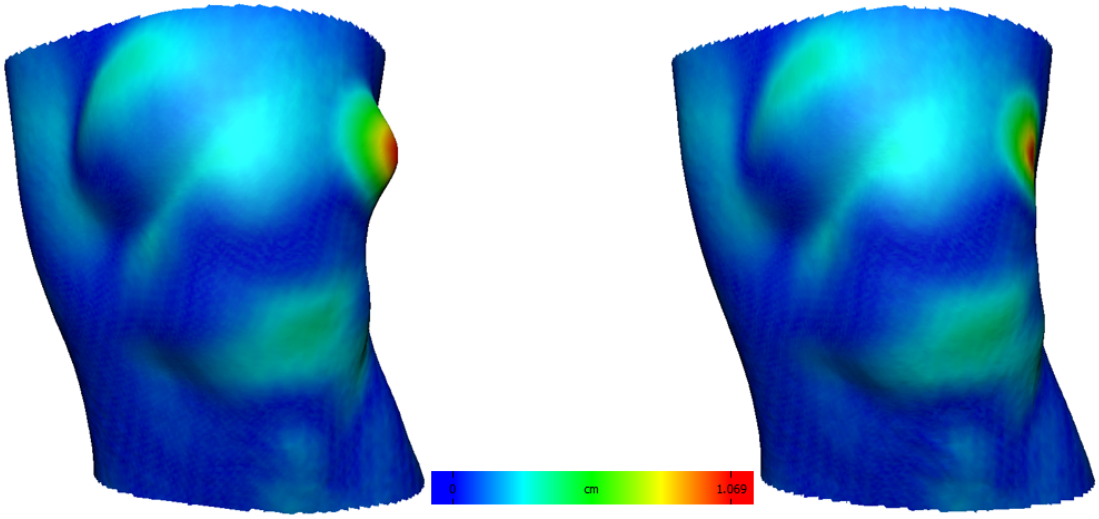

Figure 10.6: Computation of the distance for the first pair of knees, with a deformation on the left one.

The area of the left knee is 621 cm^2, while the area of the right knee is 618.6 cm^2; the difference is 2.4 cm^2.

The volume of the left knee is 1855 cm^3, while the volume of the right knee is 1847 cm^3; the difference is 8 cm^3.

The animation shows that while the changes in other zones look like if they were the result of a change of position, the deformation is not, looking very unnatural.

10.2.2 Second pair

After computing the distances, we obtain the following results, shown in figure 10.6.

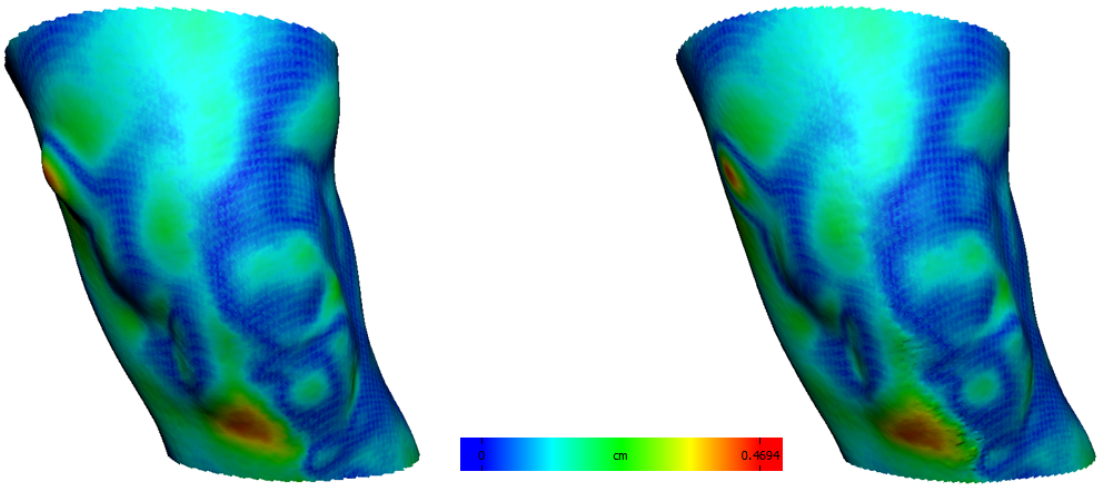

Figure 10.7: Computation of the distance for the second pair of knees, with a deformation on the left one.

The area of the left knee is 823.5 cm^2, while the area of the right knee is 903.6 cm^2; the difference is 80.1 cm^2.

The volume of the left knee is 2560 cm^3, while the volume of the right knee is 2607 cm^3; the difference is 47 cm^3.

The animation shows that the only zone that does not look like if it was affected by a change of position only is the deformation we have added, giving us the difference between the two zones that have the maximum distance, marked in red in the coloured knee.

10.2.3 Third pair

After computing the distances, we obtain the following results, shown in figure 10.8.

The area of the left knee is 754.9 cm^2, while the area of the right knee is 753.2 cm^2; the difference is 1.7 cm^2.

The volume of the left knee is 2050 cm^3, while the volume of the right knee is 2014 cm^3; the difference is 36 cm^3.

The animation shows that while there are two zones with the maximum distance, there is

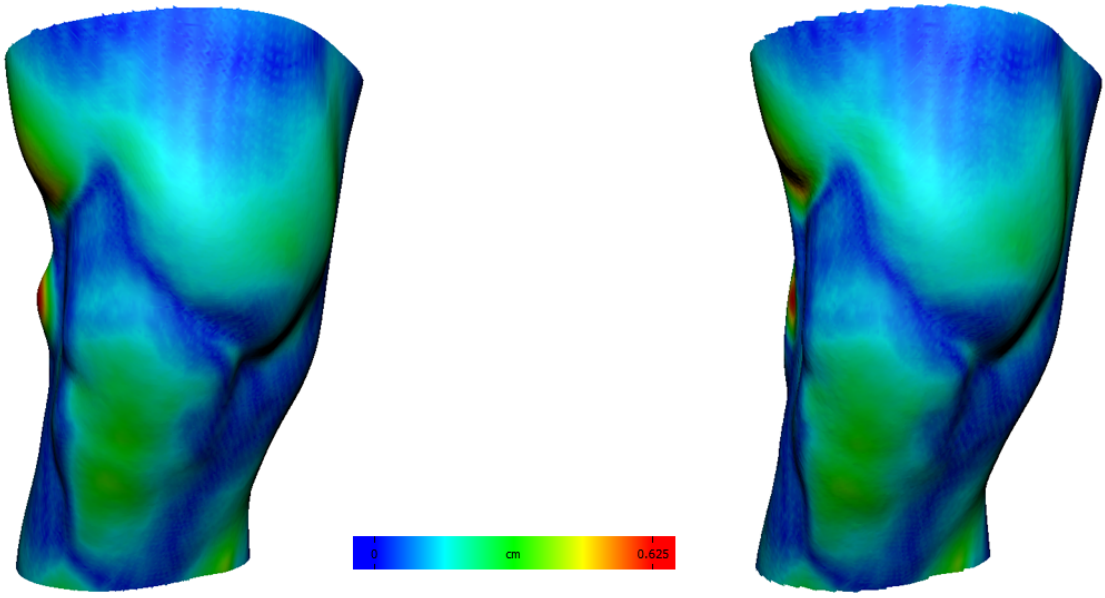

Figure 10.8: Computation of the distance for the third pair of knees, with a deformation on the left one.

one that is the result of a different position of the knee, while the other, the deformation, looks very unnatural.

10.3 Kinect test

Using our program with a model obtained with *Kinect* becomes more difficult than using a model obtained with a light-based 3D scanner. The first difficulty is the fact that with a model that is less detailed it is more difficult to get a good manual alignment. The second difficulty is a known fact, when clipping using planes we do not cut the faces if the plane crosses them, and in a model with low resolution this can lead to noticeable errors in the calculation of the volume or the area. The third one comes from the fact that the models obtained with the *Kinect* are very noisy. Nevertheless, we have used our program with the *Kinect*-scanned model.

After computing the distances, we obtain the following results, shown in figure 10.8.

The area of the left knee is 1209 cm^2, while the area of the right knee is 1174 cm^2; the difference is 35 cm^2.

The volume of the left knee is 3199 cm^3, while the volume of the right knee is 2954 cm^3; the difference is 245 cm^3.

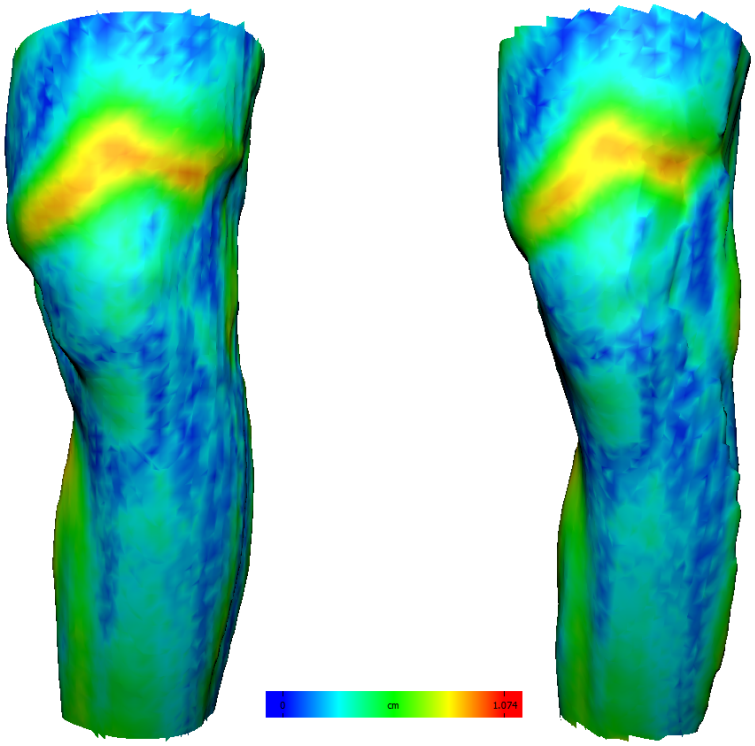

Figure 10.9: Computation of the distance for the third pair of knees, scanned by a *Kinect*.

Animation shows that the differences are distributed along the whole knee, meaning that differences are not caused by inflammation in an area of the knee.

10.4 Multi-view stereo test

In this test we have the same problems that we had when testing our program with the *Kinect*, that is, the difficulty of getting a good manual alignment, the errors that appear when clipping using planes and the errors derived from a noisy model. Again, we have done the test to see what are the results.

After computing the distances, we obtain the following results, shown in figure 10.10.

Since the scale is arbitrary, we have not included the scale in the image of the colouring, and we have not measured the area and the volume, because it would not give any valuable information.

Animation shows that the differences are distributed along the whole knee, meaning that differences are not caused by inflammation in an area of the knee. There are also many

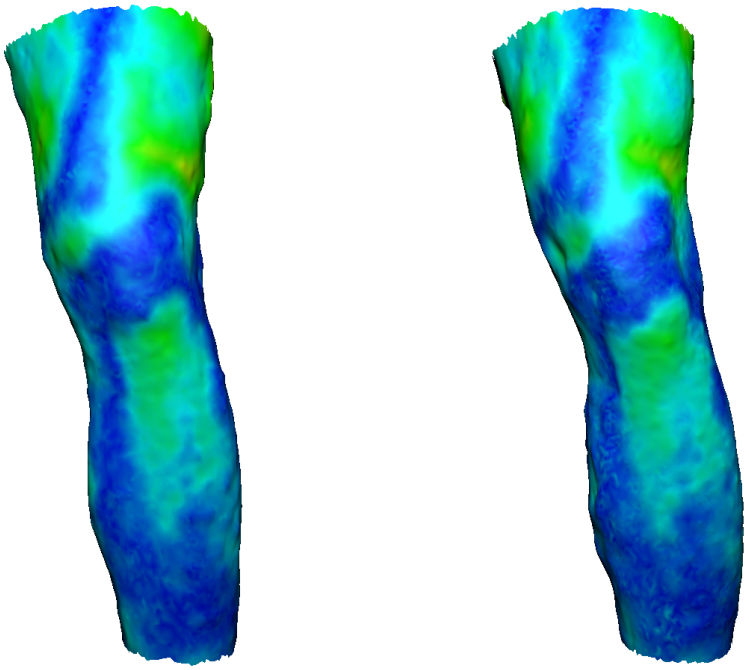

Figure 10.10: Computation of the distance for the third pair of knees, using multi-view stereo.

differences caused by noise.

10.5 Results overview

In the following lines we aim to summarise the results obtained in the previous tests.

We are going to comment the results of the comparison of left and right knees of the same person first. Since we scanned knees of people which were not suffering from inflammation, we expected the left and the right knee to be similar. However, we did not expect them to be that similar, since after computing the distances we can observe that in most of the knee the difference is around 1 or 2 mm, while the maximum is around 5 mm, which is lower than the differences that can be obtained between a relaxed and a contracted muscle. Also, with the exception of the second case, where we know that the noise makes impossible to correctly compute areas and volumes, the difference between areas is minimal and the difference between volumes is very small. So, we have observed that the degree of similarity of a left and a right knee from the same person is really high if there is no inflammation.

Now we are going to analyse the results of the comparison of left and right knees,

with the left one having a simulated deformation. First of all, the combination of the colouring based on the distance and the animation is the most useful method to check for inflammation. While the colouring gives a measure that can be result of either inflammation or contracting or relaxing the muscles, the animation shows the process of going from a position from another, thus showing us that the deformation is unnatural. About the area and the volume, they are useful in cases of an extended inflammation which covers a big area of the joint, but not in cases where it affects a small zone, since in these cases a change in the position of the joint in the scanning moment can have an effect in the value bigger than the effect caused by inflammation.

Finally, we have to comment that given the current precision of the *Kinect* and the multi-view stereo techniques, they are not useful for our purposes unless the amount of swelling is very high, since they are not precise enough and noise causes more variation in the model than a small amount of swelling would cause.

11 Final conclusions

This section aims to summarise the final conclusions that can be drawn from the development of this project. First of all, we are going to comment the conclusions drawn after reviewing the final state of the application developed. After that, we are going to comment the main areas of improvement of this project.

11.1 Conclusions

At the start of the project we presented the objectives of the project, which can be summarised as creating a tool able to compare data of the same body part in different states of inflammation and give information resulting of this comparison and given that we proposed using a 3D model of the body part as data, explore low-cost scanning techniques to obtain this kind of data.

We can say that we have fulfilled the objectives proposed. The application that we have created is able to load 3D models of body parts and compare them, giving information to the user extracted from the calculation of different measures. Moreover, we have explored three low-cost scanning techniques that can be used to obtain 3D models needed.

The comparison methods are easy to understand and they give valuable information. The most important method is based on calculating distances between the two models, showing it as a colour map, which visually shows the zones which suffer from inflammation, or as an animated transition, which allows the user to determine if the colour map obtained is caused by inflammation or by other factors like a different position in the moment that body parts were scanned. Two more methods are available, which are the calculation of the area and the volume, both giving information that, while not being always useful, can be helpful in cases where the inflammation affects a big area of the joint.

About the scanning techniques, the results are satisfying, but they could have been even better. The light-based 3D scanner gives precise results, while the *Kinect* and the multi-view stereo techniques do not have enough precision for our purposes. That means that unfortunately most of the people will have limited access to the technique that gives reliable results. However, given the increasing popularity of 3D printers, it is possible than in a near future the amount of people having access to a high quality 3D scanner

increases, knowing that, for example, the newest version of the *Kinect* offers improved precision and newer versions might come.

Apart from the aspects related to the results obtained in relation to the objectives proposed, we have to comment what has this project supposed in the technical area.

First of all, this has been the first complex application that we have developed using *Qt* and *OpenGL*, since we had already worked with them but starting with a working application and applying small modifications. This fact has led to difficulties but after developing the application we can say that we have gained enough experience to face more difficult challenges in the future.

The coding process has not been difficult since most of the code is using simple procedures, and the complex code and non standard data structures have come from an external library, the *Point Cloud Library*. The use of this library has led to the need to perform extensive tests to determine the parameters of one of the algorithms that we have used, the ICP, which has been used to align two 3D models. Studying the performance of this algorithm has given us information about the role that the user of our application has to be given in the aligning process. Also, we have taken advantage of developed tools to use low-cost scanning techniques like the *Kinect* or multi-view stereo, thus avoiding developing complex codes that could even be a project by themselves.

To sum up, the result of the project has been satisfying because we have fulfilled the objectives proposed at its start. We have developed a working application and we have explored comparison and scanning techniques with mostly positive results. In the technical area, we have used different resources related to computed graphics in order to create the program.

11.2 Future work

We can say that we have fulfilled the objectives proposed in the start of the project, but there is still room for improvements. We suggest the following areas of work:

- Create the documentation of the code and a user manual for the software.
- Enhance the software with tools that allow modifying the meshes, that is, aligning, mirroring, rotating or scaling them, in order to allow the user to adapt a 3D model to the needs of the program without the need of other software.
- Add features that improve the usability of the program, such as automatic selection of some properties like the unit used based on the unit used in the model or the calculation of clipping planes that are initially selected.

- Do extensive tests using other kinds of joints, like ankles or elbows, and use patients suffering from inflammation as test cases.
- Do further exploration of scanning techniques, using the newest version of the *Kinect*, other devices containing depth cameras and trying to improve the results obtained with multi-view stereo techniques by understanding how photographs have to be taken to get the best results.
- Try to adapt the software to mobile devices, given the increasing popularity of the use of this kind of devices for any purpose.

12 Bibliography

[1] Laura Ferrero-Miliani et al. "Chronic inflammation: importance of NOD2 and NALP3 in interleukin-1β generation". In: *Clinical & Experimental Immunology* 147.2 (2007), pp. 227–235. ISSN: 1365-2249. DOI: 10.1111/j.1365-2249.2006.03261.x. URL: http://dx.doi.org/10.1111/j.1365-2249.2006.03261.x.

[2] David L. Scott, Frederick Wolfe, and Tom W. J. Huizinga. "Rheumatoid arthritis". In: *The Lancet* 376.9746 (Sept. 2010), pp. 1094–1108. DOI: 10.1016/s0140-6736(10)60826-4. URL: http://dx.doi.org/10.1016/s0140-6736(10)60826-4.

[3] Mark de Berg et al. "Computational Geometry". In: *Computational Geometry*. Springer Berlin Heidelberg, 2000, pp. 1–17. ISBN: 978-3-662-04247-2. DOI: 10.1007/978-3-662-04245-8_1. URL: http://dx.doi.org/10.1007/978-3-662-04245-8_1.

[4] A.D.A.M. Medical Encyclopedia. *CT scan: MedlinePlus Medical Encyclopedia*. Nov. 9, 2012. URL: http://www.nlm.nih.gov/medlineplus/ency/article/003330.htm (visited on Oct. 2, 2014).

[5] Apple Inc. *Instant Heart Rate - Heart Rate Monitor by Azumio for Free featuring workout training programs from Fitness Buddy on the App Store on iTunes*. URL: https://itunes.apple.com/us/app/instant-heart-rate-heart-rate/id409625068?mt=8 (visited on Oct. 1, 2014).

[6] Khronos Group. *OpenGL - The Industry Standard for High Performance Graphics*. URL: https://www.opengl.org/ (visited on Jan. 12, 2015).

[7] Microsoft. *Direct3D - Windows app development*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx (visited on Jan. 12, 2015).

[8] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm". In: *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. 2001, pp. 145–152. DOI: 10.1109/IM.2001.924423. URL: http://dx.doi.org/10.1109/IM.2001.924423.

[9] Rachid El Khoury, Jean-Philippe Vandeborre, and Mohamed Daoudi. "3D mesh Reeb graph computation using commute-time and diffusion distances". In: *Three-Dimensional Image Processing (3DIP) and Applications II*. SPIE, Feb. 2012, 82900H–82900H-10. DOI: 10.1117/12.906724. URL: http://dx.doi.org/10.1117/12.906724.

[10] Mark Foskey, Ming C. Lin, and Dinesh Manocha. "Efficient computation of a simplified medial axis". In: *Journal of Computing and Information Science in Engineering* 3.4 (2003), pp. 274–284. DOI: 10.1115/1.1631582. URL: http://dx.doi.org/10.1115/1.1631582.

[11] Avneesh Sud, Miguel A. Otaduy, and Dinesh Manocha. "DiFi: Fast 3D Distance Field Computation Using Graphics Hardware". In: *Computer Graphics Forum* 23.3 (2004), pp. 557–566. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2004.00787.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2004.00787.x.

[12] Paolo Cignoni. *MeshLab*. URL: http://meshlab.sourceforge.net/ (visited on Nov. 25, 2014).

[13] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[14] Artec Group. *Artec 3D Scanners*. URL: http://www.artec3d.com/ (visited on Oct. 2, 2014).

[15] Microsoft. *Kinect for Windows*. URL: http://www.microsoft.com/en-us/kinectforwindows/ (visited on Oct. 1, 2014).

[16] Microsoft. *Kinect Fusion*. Sept. 11, 2014. URL: http://msdn.microsoft.com/en-us/library/dn188670.aspx (visited on Oct. 1, 2014).

[17] Steven M. Seitz et al. "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. June 2006, pp. 519–528. DOI: 10.1109/CVPR.2006.19. URL: http://dx.doi.org/10.1109/CVPR.2006.19.

[18] Inc. Autodesk. *Autodesk 123D Catch — 3d model from photos*. URL: http://www.123dapp.com/catch (visited on Oct. 3, 2014).

[19] *Object Files (.obj)*. URL: http://www.martinreddy.net/gfx/3d/OBJ.spec (visited on Nov. 25, 2014).

[20] David Eberly. *Polyhedral mass properties (revisited)*. Dec. 31, 2002. URL: http://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf (visited on Jan. 14, 2015).