The Visualization process applied to the medical field

Michael Pierre DUPILLIER - VICOMTech Internship

February 26, 2008

0.1 Acknowledgment

First of all I would like to thank Julián Flórez and Jorge Posada, respectively general manager and assistant director of the research center VICOMTech, for welcoming me for my internship.

Special thanks to Céline Paloc, the responsible of the biomedical department, for her cordial welcome, and for the opportunity she gave me to enter VICOMTech.

I also owe a special debt of gratitude to the VICOMTech team for the enjoyable atmosphere they created and for their generosity with their time and expertise.

And finally I would like to thank Iván Macía, my supervisor, who taught me a lot and never hesitated to help me.

Résumé

Cela fait peu de temps que la 3D peut être utilisée comme un outil efficace dans les applications médicales. Jusqu'à il a quelques années, les microprocesseurs et les cartes graphiques ne possédaient pas la puissance nécessaire pour reconstruire des image médicales en 3D sur un moniteur. Le domaine médical est donc longtemps resté réticent à l'utilisation de telles technologies. Mais avec l'augmentation des connaissances et de la puissance des machines actuelles, l'utilisation de la 3D semble devenir un moyen efficace et rapide pour interpréter les données médicales.

Le stage effectué au sein de l'entreprise VICOMTech m'a permis de découvrir le process de visualisation au travers le développement d'une application (et de plusieurs prototypes) dédiés a la simulation et la préparation d'intervention chirurgicales pour la mise en place de prothèses dentaires.

Entouré par une equipe dynamique et qualifiée, ce stage m'a permis d'apprendre beaucoup sur la réalisation de logiciels dans un environnement concurrentiel et dans domaine tres interessant qu'est la vizualisation 3D.

Contents

List of Figures

Chapter 1

Introduction

1.1 Purpose of the report

This document presents a summary of the work accomplished during the five and a half monthss spent on TN09 internship in the VICOMTech research center. To be comprehensible both by the concerned persons at the UTC and by my work team in VICOMTech, this document has been written in English. A summary has also been written in french for the people of the UTC who do not understand Shakespear's language. The training period ran through the autumn semester (A07) from the 3rd of September 2007 to the 15th of February 2008. As advised by the UTC, this document will not explore technical questions, but will explain the different developments, the issues met and the ways used to solve them. Some pieces of code are displayed in this report only in order to illustrate and to explain better some parts of the accomplished work in VICOMTech. This document must not be diffused without VICOMTech authorization.

1.2 VICOMTech Presentation and Work Team

1.2.1 VICOMTech

VICOMTech (Visual Communication and Interaction Technologies Centre) is an applied research centre for Interactive Computer Graphics and Multimedia located in the Technology park of San Sebastian. VICOMTech is a non-profit association, founded by the INI-GraphicsNet Foundation, with the Fraunhofer-IGD as founder member, and the Basque Television, Radio and Broadcasting group EiTB.

VICOMTech works in applied research in the following application areas:

- Digital TV and Multimedia Services
- Biomedical Applications
- Tourism, Heritage and Creativity
- Interaction for Education, Leisure and E-Inclusion
- Industrial Applications

The Biomedical Applications Department of VICOMTech carries out research and development for the healthcare and biotechnology sectors. I did all my internship in this area. It is important to note that VICOMTech is a non profit organisation. This mainly means that VICOMTech is not allowed to commercialize any final product. VICOMTech works for other companies or state organisations.

1.2.2 Work Team

Department Head:

- Céline Paloc, Dr. in Biomedical Engineering

Researchers:

- Iván Macía Oliver, Ingeniero Industrial, Ingeniero en Automática y Electrónica Industrial
- Iñigo Barandiaran Martirena, Ingeniero en Informática
- Diana Wald , Ingeniera en Informática

1.2.3 Internship Presentation

The internship mainly dealt with the vizualisation and the processing interaction of medical images, but the knowledge required (or acquired during the internship) was more generaly attached to computer graphics and software design.

The Biomedical departement of VICOMTech, investigates and carries out some other projects for other corporations. As part of the confidentiality agreement I signed with VICOMTech, I will not divulge the name of any corporation VICOMTech is working for. However, during my internship I have been involved in the developpement of an application dedicated to the simulation and the planning of dental surgery. To understand better the report, the corresponding client of VICOMTech will be mentioned under the alias of IMP. IMP designs and manufactures dental implants, prosthetic components and surgical instruments. VICOMTech and IMP signed a contract according to which VICOMTech develops a "software" specialized in the planning of implants for individual surgeries.

The software term personalized has quoted VICOMTech as a non profit organisation. VICOMTech cannot sell any final product but investigates and creates prototypes for other corporations. As a result, the product VICOMTech provides can be considered both as an investigation project and also as an efficient and safe prototype.

As far as the context of the project is concerned, IMP already had an application which allowed the users to display a medical image in 3D, insert some implants, modify them etc... but the 3D interfaces were not well defined, were slow and not easy to use. Moreover, IMP was not the owner of the source code so no modifications could be done.

In the first part of this report will be an introduction to the vizualisation process assisted by computers. Then two different techniques of vizualisation (Surface rendering and Volume Rendering) will be explained through the work I had to realise during my internship.

Chapter 2

The Vizualisation Process

2.1 Introduction

By definition, Visualization is any technique for creating images, diagrams, or animations used to communicate a message. Visualization through visual imagery has been an effective way to communicate both abstract and concrete ideas since the dawn of man. Examples from history include cave paintings, Egyptian hieroglyphs, Greek geometry, etc. Maybe the most common example of visualization assisted computers, by corresponds to the digital animations produced to present meteorological data during weather reports on television, though few can distinguish between those models of reality and the satellite photos that are also shown on such programs. TV also offers scientific visualizations when it shows computer drawn and animated reconstructions of road or airplane accidents.

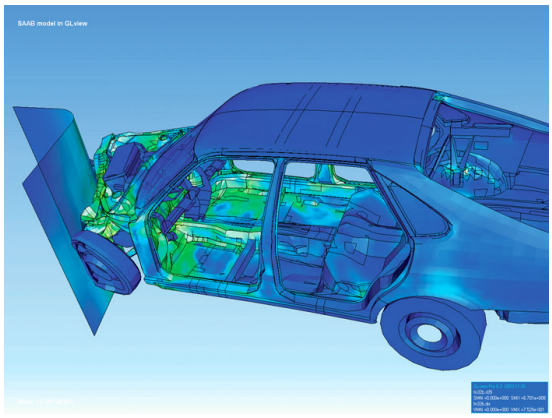

Figure 2.1: Visualization of how a car deforms in an asymmetrical crash using finite element analysis (Visualisation de la deformation assymetrique de la carroserie d une voiture lors d un crash)

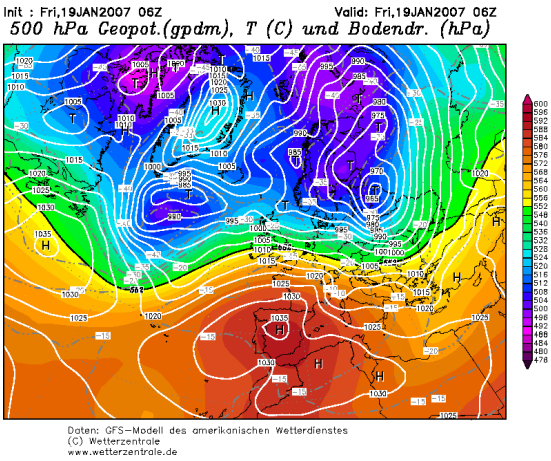

Figure 2.2: Visualisation of the atmospherical pressure using blended color gradient and isoline representation (Visualisation des pressions atmospheriques a l aide d un gradient de couleur et d iso-lignes)

2.2 The Visualization Pipeline

Before going further the main concept attached to the visualization process must be introduced. Visualization transforms data into images that efficiently and accurately convey information about the data. The visualization pipeline corresponds to all operations that transform one (or more) input data into one (or more) output image. Inside this pipeline, the data is transformed, from a type to another then to an other etc, until the end of the pipeline. Each modification of the data inside the pipeline is done using a filter. This important notion is illustrated on the figure bellow.

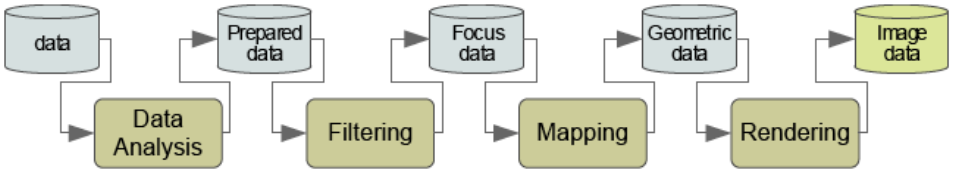

Figure 2.3: Visualization Process. (Process de vizualisation

1. Data Analysis: data is prepared for visualization (e.g., by applying a smoothing filter, interpolating missing values, or correcting erroneous measurements) – usually computer-centered, little or no user interaction.
2. Filtering: selection of data portions to be visualized – usually user-centered.
3. Mapping: focus data are mapped to geometric primitives (e.g., points, lines) and their attributes (e.g., color, position, size); most critical step for achieving Expressiveness and Effectiveness.
4. Rendering: geometric data are transformed to image data.

As it will be explained later, my work here in VICOMTech required using all the different parts of the visualization pipeline.

2.3 The Visualization Toolkit

2.3.1 Presentation

In this subsection will be introduced the Visualization toolkit. The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics, image processing, and visualization. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging / 3D graphics algorithms and data. VTK integrates also its own default window, and user interaction system.

2.3.2 A visualizaton oriented software system

The strenth of VTK resides in the fact that the implementation jibe directly to the visualization pipeline seen in the introduction of this section. The following example shows the source code written to display the contours of the implicit quadric function in (Figure - 3.2).

```
1 package require vtk
2 package require vtkinteraction
3
```

```
#creates a mathematic quadric function
vtkQuadric quadric
  quadric SetCoefficients .5 1 .2 0 .1 0 0 .2 0 0

# vtkSampleFunction samples an implicit function over the x-y-z range
# specified (here it defaults to -1,1 in the x,y,z directions).
vtkSampleFunction sample
  sample SetSampleDimensions 30 30 30
  sample SetImplicitFunction quadric

# Create five surfaces F(x,y,z) = constant between range specified. The
# GenerateValues() method creates n isocontour values between the range
# specified.
vtkContourFilter contours
  contours SetInput [sample GetOutput]
  contours GenerateValues 5 0.0 1.2

#Map the contours into geometric primitives
vtkPolyDataMapper contMapper
  contMapper SetInput [contours GetOutput]
  contMapper SetScalarRange 0.0 1.2

#set a vtkActor
vtkActor contActor
  contActor SetMapper contMapper

# The usual rendering stuff.
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

ren1 AddActor contActor

iren AddObserver UserEvent {wm deiconify .vtkInteract}
iren Initialize

wm withdraw .
```

From the previous code it is easy to extract the pipeline. We can consider that the input is the quadric source. Then this source is passed through an implicit function filter which generates an output used as an input in the contour filter. The contour filter output is then set as the input to the mapper and finaly the mapper is saved in a drawable instance (vtkActor) which is rendered on the screen.

2.4 The Insight Toolkit

The Insight Toolkit (ITK) is an open-source software toolkit for performing registration and segmentation. Segmentation is the process of identifying and classifying data found in a digitally sampled representation. Typically the sampled representation is an image acquired from such medical instrumentation as CT orMRI scanners. In the vizualisation process, ITK can be used in the first step of the filter, ITK proposes an efficent way to load, smooth, modify or just read the data. All the filters performed on an Image "should" be done using ITK. ITKŠs C++ implementation style is referred to as generic programming, which means that it uses templates so that the same code can be applied generically to any class or type that happens to support the operations used. Such C++ templating means that the code is highly efficient, but is also, on first approach hard to control well. In the Annex of this report has been included a class I had to develop using ITK during my internship.

2.5 Visualisation Applied to the medical field

The 3D has been recently seen as a useful tool in medical application. Until a couple of years ago, graphical cards and micro processors were not fast enough to display interesting 3D reconstructions of aquired data on the screen. For this reason the medical field has been reticent to the use of 3D. But with the increasing computer power and knowledge the use of 3D has become a new way to interpret medical datas.

3D Medical Images is an organised block of data which can be acquired with a computed tomography (CT) or with a magnetic resonance imaging (MRI) scanner. Both acquisitions provide a series of grey scale cross-sectional slices. Each of these slices are stored into a single 2D image file.

The (Figure - 2.4) shows an easy way to understand and to represent the medical Image in 3D. Each pixel of the structure has a grey scale value. the all image is structured with the same number of bytes allocated to a pixel (ex: char, short, unsigned short...)

Such a structure can be visualized in different ways. The first and the most simple way is to look at each slice (image), one after the other, as a 2D image and this gives an idea of what the volume looks like (Figure - 2.5)

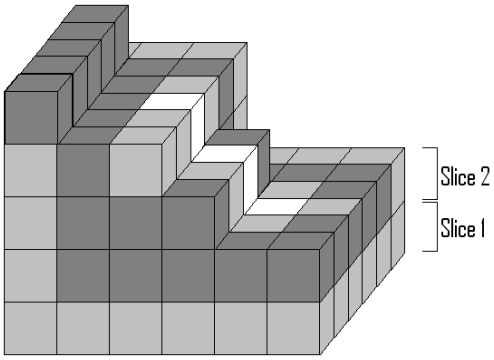

Figure 2.4: Medical image voxels Vizualisation, (Visualisation des voxels d une image Medicale)

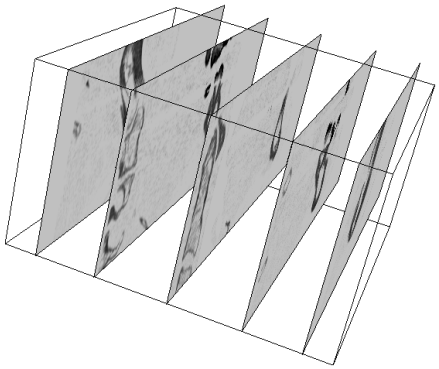

Figure 2.5: Visualization of several slices using the software developed for IMP, (Vizualisation de plusieures coupes a l aide du software developpe pour IMP)

Another method (explained in the fourth chapter) consists in visualizing the full volume in 3D using the ray cast method. The third, and not the last possible method, will be explained in the next chapter. It consists in extracting one or more surfaces from the block of data.

Chapter 3

Surface Visualization in Medical Imaging [One Month]

3.1 Introduction

As it has been said in the previous section, vizualisation is the process of generating an image from a model. When the use of a computer is needed, the visualization process is called Rendering. The model is a description of three dimensional objects in a strictly defined language or data structure. It contains geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image.

Surface rendering means that an object is stored in terms of its surface. In other words, a sphere is represented by its surface; a chair is represented by a collection of planes, cylinders etc. Surfaces are what we see, so they are a natural choice for graphical applications. Usually we use a mesh representation which consists of a (large) number of flat polygons, often triangles, that map out the surface in question. Each polygon is flat, so this will be an approximation to a curved surface, but it is easy to adjust an approximation to requirements simply by making the polygons smaller [ch2-surf-models.pdf]. This form dominates the PC games card market, making it a cost-effective form easy to render quickly.

3.2 Iso-Surface extraction

The iso-surface extraction method is used in the medical field to vizualise medical images. As illustrated in the name, the iso-surface extraction extracts a single mesh surface from a 3D image using a single scalar value.

It is useful to compare the Iso-surface extraction method with 2D contouring. More than a comparison, one can say that contouring in the 2D field corresponds to the isosurface extraction method in the 3D field.

There are some well known and common examples of 2D contouring are really common. For example, weather maps are annotated with lines of constant temperature (isotherm) or pressure (isobar). Another exemple is found on topological maps where lines of constant elevation are drawn. 3D contours are called isosurfaces.

The contour algorithm filter for both the 2D and the 3D fields works almost identically. An interpolation of the position of each isovalue point is computed on each space direction. The mesh is then reconstructed from these points.

The following images show on the left an IGN map displaying some iso-elevation lines (2D contouring) and on the right a multi iso-surface contouring extracted from a 3D volumetric function.

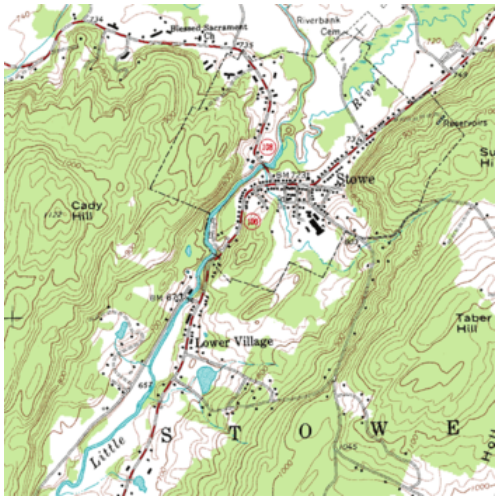

Figure 3.1: IGN Topographic Map Example, (Example de Carte topographique IGN)

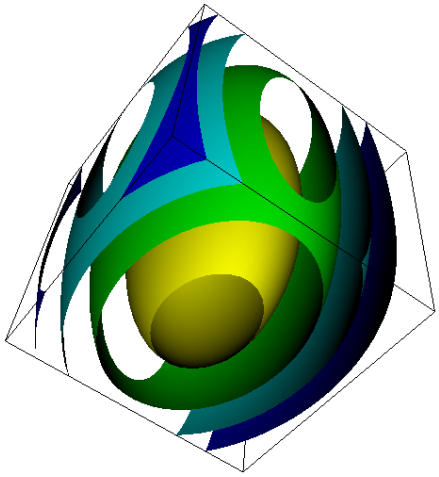

Figure 3.2: Visualization of a Quadric Function using Tcl and Vtk, (Vizualisation d une fonction quadrique en utilisant Vtk et tcl)

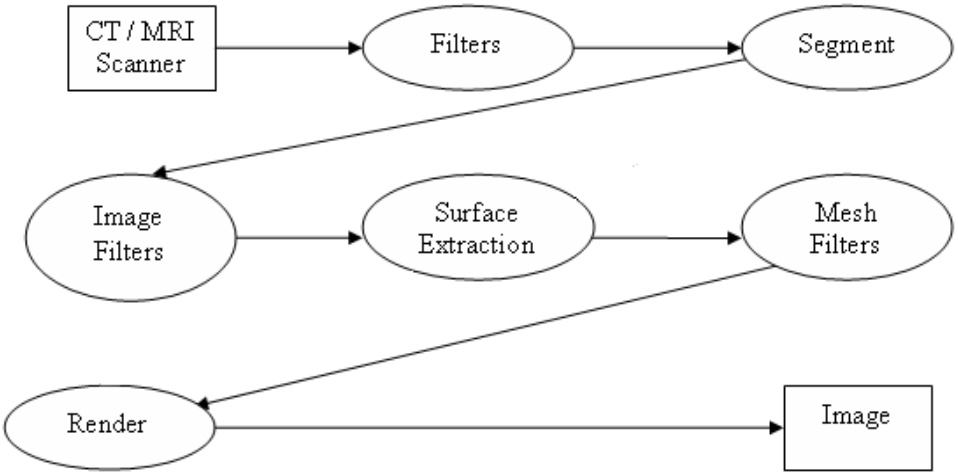

Figure 3.3: Data flow diagram corresponding to a medical image vizualisation, (Diagramme du flux de donnees correspondant a la vizualisation d une image medicale)

The (Figure - 3.3) presents the different steps of medical image visualization process. The Segmentation step corresponds to the extraction of the main information of the image. In our case, the main information to extract the mandible and the teeth. In a CT file the scalar value stored can be classified. For example the scalar values between two ranges corresponds to the bones, the scalar values between two other ranges can be considered as fat etc. In fact, there is a gradual transition between the different fields.

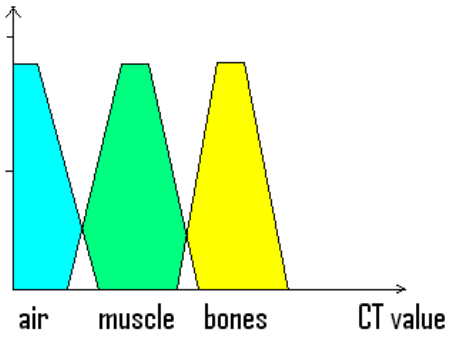

Figure 3.4: Air Muscle and Bone gradual scalar repartition in a CT Dataset,(Repartition graduelle des valeurs scalaires dans une image CT

Additional filters can be applied onto the image before its extraction. A full part of this report could be dedicated to the marching contour algorithm used to extract the surface from a 3D scalar field but the idea is to find automatically the grey scale value corresponding to the interface between the teeth and an other tissue. Finally the surface is reconstructed using the generated interface. A more detailed description of the extraction algorithm has been included in the Annex of the report.

3.3 Improving Generated 3D Models

3.3.1 Context

The following section describes the first task I was in charge of for VICOMTech. Because this task had been given to me at the beginning of my internship, I first had to learn all the concepts described before to taking ITK, VTK and Qt in hand and I then began the work described above.

As explained in introduction, VICOMTech has been involved in the development of an application dedicated to the simulation of implant surgery. This software uses a surface rendering method to render the topology of the bones and the teeth. VICOMTech asked me to search and test smoothing filters to apply on the data in order to come closer to the real surface and reduce the artefacts produced by the segmentation. Because the data used as a first step in surgery, close attention must be paid to the results, and the aim of the research was not to have a nice looking result but a result closer to reality.

3.3.2 Content of the work

I enclosed in this report the document I wrote for VICOMTech. Here is the abstract:

In this document are set forth experiment results done with ITK (Insight ToolKit) and VTK (Visualization Toolkit) in order to obtain a fast and efficient way to smooth surfaces extracted from segmented medical data. Because of the internal structure of the data and also because of the segmentation process, the rendered image presents some non desirable noises. We can consider that it is not always desirable to exactly interpolate surface data when the data are contaminated with noise. A smooth approximation is often more appropriate and even more closer to the realistic surface.

The enhancement of the Original surface can be computed by two different ways. The first way (first part of this document) is to smooth the medical image before the surface extraction by applying some filters on the image data. The second way to smooth the surface is to apply filters onto the extracted meshes (second part of this document).

The main difficulty when one deals with signals extraction and signal filtering, is to prevent the loss of important information. That's why we need to know exactly which are the important information in the signal and how high the approximation can be. As it has been written a few lines above, the ambit of our research corresponds to the reconstruction of segmented medical data. In consequence the input data we have is already approximated from the aquisition data due to the segmentation process. The aim of the filters we are searching for, is to come closer to the real surface and reduce the artefacts produced by the segmentation and not only to have a nice looking result.

3.3.3 Experimentation Methods

The efficiency of a smooth filter depends on three parameters:

- The precision of the filter
- The time used by the filter
- The resulting global aspect of the mesh

The second and the third parameters are easy to evaluate. The rendering time can be directly obtained using a timer inside our c++ code or using the ITK and VTK timers, and the global aspect of the resulting mesh can be estimated visually.

The precision of the filter can't be calculated easily. A good way to evaluate it would have been to calculate the average of the Euclidian distance between the original extracted surface and the smoothed one but neither ITK nor VTK provide such a tool.

As a consequence the precision had to be judged visually. In order to estimate the precision of the applied filter, we rendered a single contour of the smoothed surface and the corresponding single contour of the original generated surface. The way to obtain this kind of iso-surface is the following:

First a iso contour filter is applied to the dataset selecting a valid isovalue in order to obtain the surface. Then an elevation filter is applied on a single axis in order to divide the surface into different scalars slices. Finally a contour filter is applied selecting one of the generated iso-scalar lines.

The elevationFilter is a filter used to generate scalar values from a dataset. The scalar values lie within a user specified range, and are generated by computing a projection of each dataset point onto a line. The line can be oriented arbitrarily.

Figure 3.5: Experimentation Pipeline, (Pipeline de visualization de l'experimentation)

We can consider that a smoothing filter becomes better and more precise when the average distance between the original reconstructed surface and the smoothed reconstructed surface decreases. To be precise, the filter should generate an "inside/out" surface from the original surface.

I enclose in the Annex of the report the document I wrote for VICOMTech.

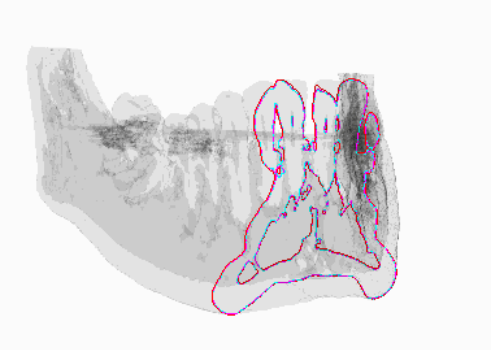

Figure 3.6: Three isolines and a blended mandibule model, (Vizualisation de trois iso-lignes et d une reconstruction semi transparente d' image CT)

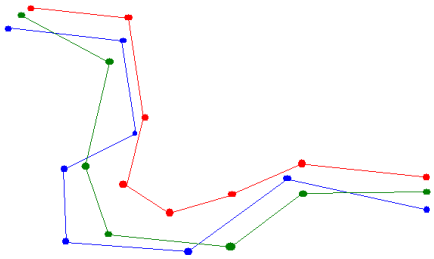

Figure 3.7: Three isolines extracted from three different surfaces. In blue is represented the isoligne extracted from the original surface, in green is represented a precise approximation of the original isoline and in red a bad approximation of the original surface, (Trois iso-lignes extraites de trois surface differentes. En bleu est representé l' isoligne de la surface originale, en rouge est representé une approximation precise de l' isoligne originale et en bleu est representé une mauvaise approximation de l iso-ligne originale)

Chapter 4

The IMP prototype [One Months]

4.1 Introduction

As mentioned in the introduction, IMP, the corporation VICOMTech worked for, already had a software able to display the mandibules in 3D and to manage the implants. But the 3D interactions and the Renderer itself were not well designed and were not easy to use. So, IMP wanted VICOMTech to develop a small prototype able to show what could or could not be done in 3D. Thus, when I finished writing the document about rendering improvement which took me about one month, VICOMTech asked me to develop a prototype able to show what could be done in 3D when using both the interaction and the mandibule and teeth reconstruction. By this time, I was able to use VTK, ITK, Qt and the Visual Studio .NET environnement

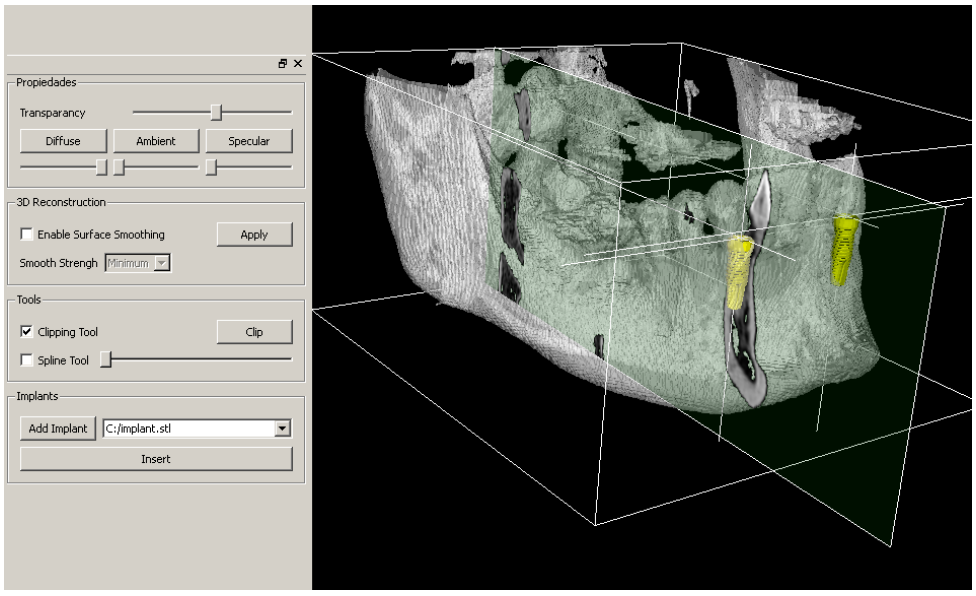

Figure 4.1: 3D Prototype preview, (Apercu du prototype 3D développé pour IMP)

4.2 Specifications

Even if the needs were not well defined, the main specification expected and which I implemented, are the following Import modules :

- T1.2 import of meta data image file

- T1.3 import of STL object (Implants)

Vizualisation 3D Module;

- T2.6 Visualization of the reconstructed Object
  - Visualization of transparency level
  - Visualization in false colors
- T2.7 Visualization of Implants
  - Visualization of transparency level
  - Visualization in false colors

3D Reconstruction Module

- T3.4 Import of the loaded meta image file
- T3.5 IsoSurface Generation
- T3.6 Surface smoothing

Interaction Module:

- T4.12 Navigation
  - Pan.
  - Zoom
  - Rotation.
- T4.13 Selection of 3D objects
- T4.14 Modification of the properties
  - Modification/Selection of the transparency value
  - Modification/Selection of the false color
- T4.17 Picking.
- T4.18 Implant insertion

4.3 Result

The application was developped using the Qt library in the VisualStudio .NET environnment. The prototype was developped in C++ and the CMake compiler was used to build it. The 3D interactions has been set using the vtk Interaction system and the 3D Renderer by using the VTK library.

The following images illustrate the implemented tool of the prototype.

The clipping tool corresponds to a crop in the dataset, displayed onto a plane at the corresponding position. In the Annex of this document will be explained not only how to display this plane but also how to color the image extracted from the dataset using a lookUpTable.

The spline tool has been designed to interactively positionate a line corresponding to the position of the dental nerve of the inferior mandibule. The surgeon needs to know where the nerve is, before leating the implant.

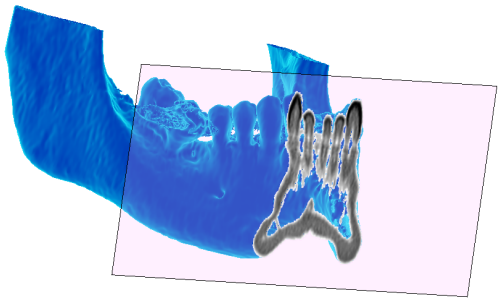

Figure 4.2: Interactive clipping Tool that allows a single slice vizualisation. (Outil de coupe interactif premettant de vizualiser un plan de coupe de l image medicale)

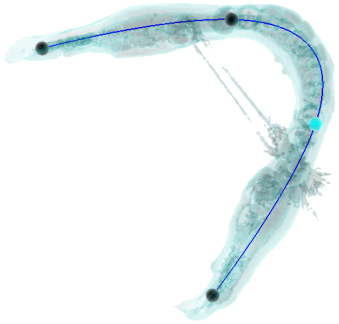

Figure 4.3: Spline Tool that allows the user to define the position and the form of the dental nerve using the mouse interactions, (Outil permettant la mise en place interactive du nerf dentaire sur le modele 3D)

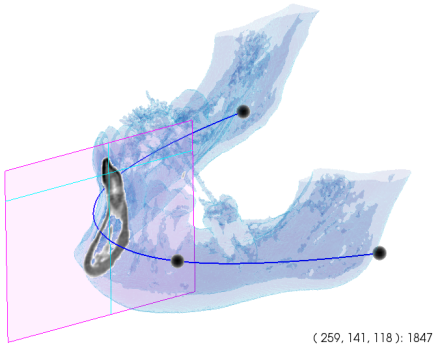

Figure 4.4: Combined Tool allowing the Clipping tool to follow the dental nerve, (outil combiné permettant de déplacer le plan de coupe le long du nerf dentaire

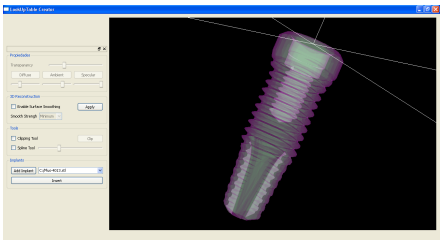

Figure 4.5: Implant Tool

The combined tool corresponds to a combination of the cut plane and the spline tool. The user can move the cut along the spline interactively with it.

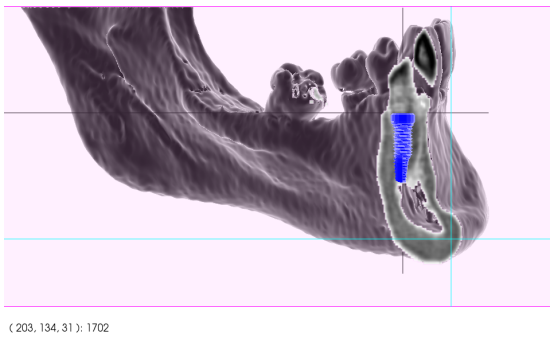

Figure 4.8: Implant interactive postition definition, (Mise en place interactive des implants en 3D avec la souris)

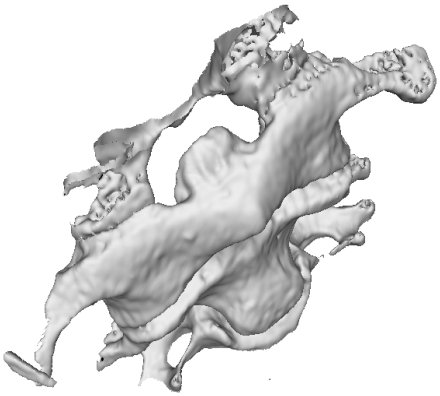

Figure 4.6: Smoothing options; Full Smoothed model, (Option d optimisation de la surface, surface optimisee au maximum)

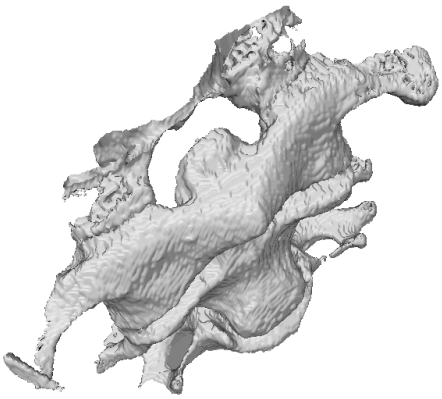

Figure 4.7: Smoothing options (non smoothed model), (Surface originale non optimisee)

The 3D interactions such as rotation, zoom, span etc, were implemented on the mouse events. The clipping tool and the spline tool can also be modified (ie move rotated ...) using the mouse interaction. Between others, the prototype implement some lighting, smoothing, blending and coloring options visible on the above figures. Finally, the menu and the user interface were developped using the Qt library.

Chapter 5

Volume Rendering [One month]

5.1 Introduction

For many applications such as walk-throughs or terrain visualization, drawing geometric primitives to visualise the information is obviously the most efficient and effective representation for the data. In contrast, some applications require us to visualize data that is inherently volumetric (which we refer to as 3D image or volume datasets). For example, in biomedical imaging, we may need to visualize data obtained from CT or MR scanner. As a result of the popularity and usefulness of volume data over the last several decades, a board class of rendering techniques known as volume rendering has emerged. The purpose of volume rendering is to effectively convey information within volumetric data. In this Section, I will introduce the basic concepts and theory of volume rendering. A second part will be dedicated to the work done using the volume rendering technology

5.2 Volume Rendering Concept

A variety of methods exist in volume rendering, but since I only had to work with the Image-Order Volume Rendering, I will not explain the other methods in this document. Image-Order volume rendering is often referred to as ray casting or ray tracing. The basic idea is that we determine the value of each pixel of the screen by sending a ray through the pixel into the scene according to the current camera parameters. We then evaluate the data encountered along the ray using some specified functions in order to compute the pixel value.

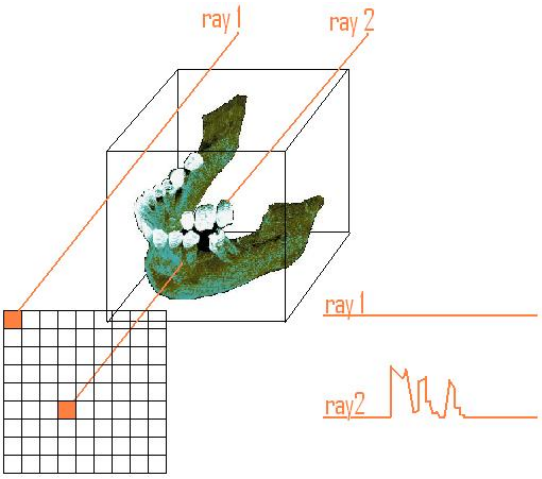

Figure 5.1: Ray-Casting method illustrated with the vizualisation of a jaw medical image. (Methode du ray casting illustree par la vizualisation d une mandibule humaine))

The ray casting process is illustrated in the (Figure - 5.1). The ray 1 does not meet any object so the generated signal is null. The second ray goes through the volume and meets some values and so the signal is not null. The two main steps of ray casting determine the values along the ray, and then process these values according to a ray function. In implementation these two steps are typically combined.

5.3 The task of finding a good ray function

5.3.1 The transfer function model

Each ray sent into the volume is sampled at regular intervals throughout the volume. The data is interpolated at each sample point, the transfer function is applied to form an RGBA sample, the sample is composited onto the accumulated RGBA of the ray, and the process repeated until the ray exits the volume. The RGBA color is converted to an RGB color and is deposited in the corresponding image pixel.

There are different way to compose the sample into the accumulated RGBA of the ray.

The first one called maximum intensity projection or MIP is a computer visualization method for 3D data that projects in the visualization plane (screen) the voxels with maximum intensity that fall in the way of parallel rays traced.

This technique is computationally fast, but the 2D results do not provide a good sense of depth of the original data.

5.3.2 The dataset Histogram

In order to simplify the task of manually finding an interesting transfer function, the scalar value repartition histogram can be displayed. (example fig 5.3)

If we consider a discrete greyscale image, and let n be the number of occurrences of grey level i. The probability of an occurrence of a pixel of level i in the image is

$p(x_i) = \frac{n_i}{n}, i \in 0, ..., L-1$

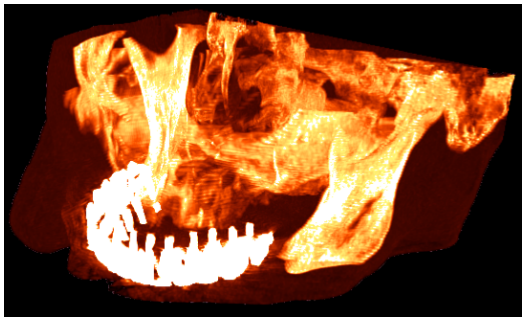

Figure 5.2: A Maximum intensity projection of a CT Head Dataset created with a ray casting method (image created with the volume rendering prototype), (Visualisation d un crane en utilisant la projection de l intensite maximale, (image rendu avec le prototype de rendu volumique))

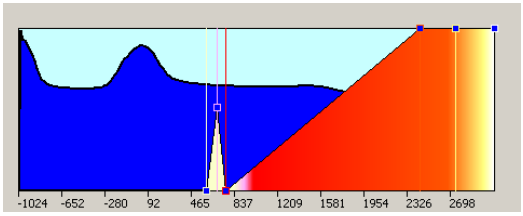

Figure 5.3: Corresponding Color and Opacity Histogram repartition, (image created with the volume rendering prototype), (Histogramme de la couleur et de l opacite correspondant a la visualisation du crane. (image cree dans le prototype de rendu volumique))

L being the total number of grey levels in the image, n being the total number of pixels in the image, and p being in fact the image's histogram, normalized to $[0, 1]$.

Going from a volumetric CT data to a 1D histogram can be considered as a filter applied to the original data to obtain an histogram output. The vizualisation toolkits such as VTK or ITK directly implement these filters.

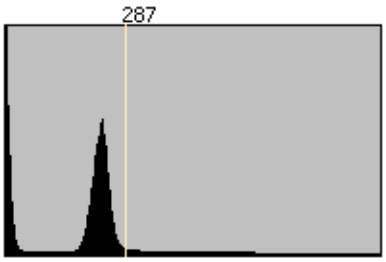

Figure 5.4: 1D histogram of a CT Dataset, (Histogramme 1D reconstruit à partir d'une image medicale)

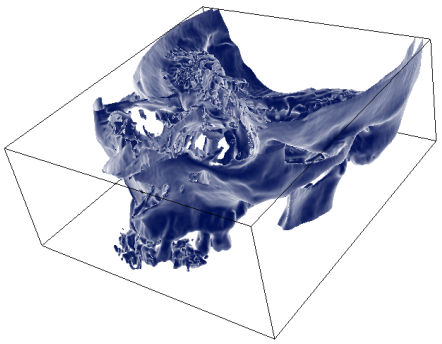

Figure 5.5: Corresponding CT reconstruction using marching contour algorithm with the isovalue 287, (Surface reconstruite à partir de la valeur 287)

Once the histogram is created, the idea is to overlay interactively a 2D linear plot on the histogram. The alpha value is represented on the Z axis, and each point of the plot can store a RGB color.

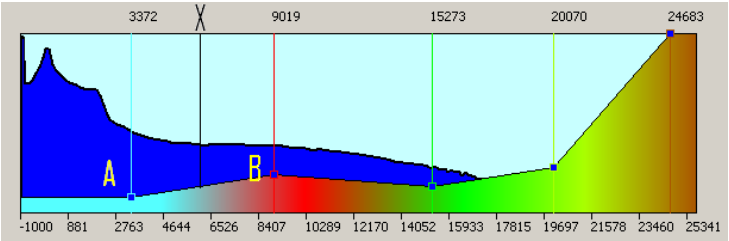

Figure 5.6: Histogram with a Linear alpha and RGB plot, (Histogramme d' une image medicale avec un gradient RGB)

Finally, the RGBA values set on the 2D overlayed plot are be mapped on the CT data scalar field. Considering the two points A and B and the unknown RGBA value of the X point:

$$r_x = (X - A)/(B - A) * (r_B - r_A) + r_A$$
$$g_x = (X - A)/(B - A) * (g_B - g_A) + r_A$$
$$b_x = (X - A)/(B - A) * (b_B - b_A) + r_A$$
$$a_x = (X - A)/(B - A) * (a_B - a_A) + r_A$$

5.4 The volume rendering prototype

The idea was to develop an application which could help the user to easily find some efficient transfer function to visualize CT, or Meta image data. A non exaustive list of the operations implemented in the prototype is the following:

- Open a CT Image or a Meta Image File and display it on the screen using the rayCasting method (figure 5.7, point 1)
- Create manually some transfer function using the scalar values repartition histogram (figure 5.7, point 6)
- Display a 2D image slice of the Original data and go through the Image (figure 5.7, point 2)
- Set the postRayCasting method (Maximum to Intensity Projection or Composite mode) (figure 5.7, point 3)
- Set the illumination properties of the model (figure 5.7, point 4)
- Load, Save a created transfert function plus the illuminations and raycasting mode (figure 5.7 point)
- Crop the volume using an interactive 3D Box Widget. (figure 5.7 point 3)

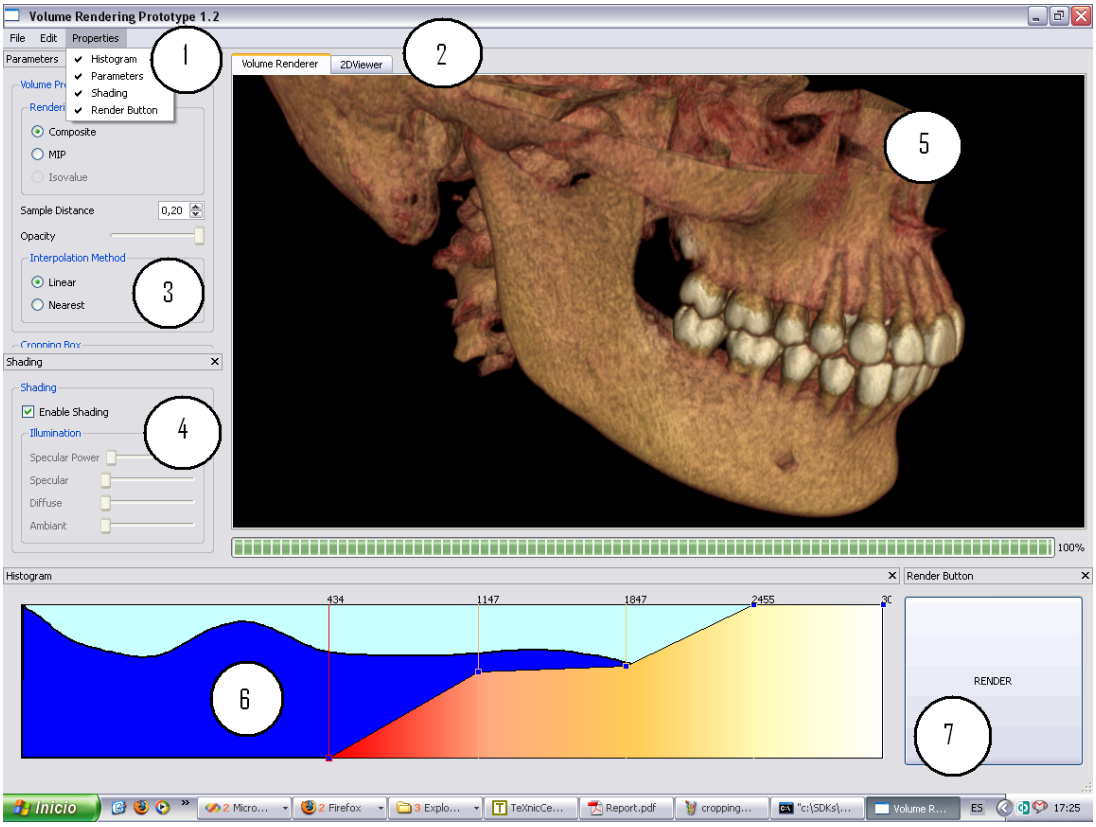

Figure 5.7: Volume Rendering Prototype Overview. (Appercu du render de Volume)

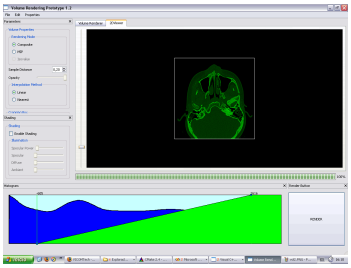

Figure 5.8: Prototype Overview [1](The 2D Image Plane Widget and the Histogram Widget), (Appercu du prototype, (Outil de vizualisation d' image en 2D plus l' outil de creation d' histogramme manuel))

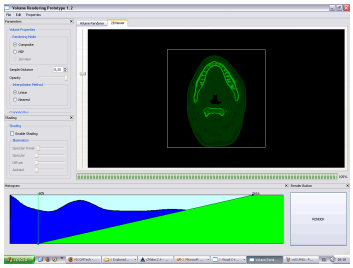

Figure 5.9: Prototype Overview [2](The 2D Image Plane Widget and the Histogram Widget), (Appercu du prototype, (Outil de vizualisation d' image en 2D plus l' outil de creation d' histogramme manuel))

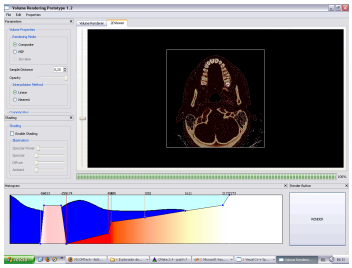

Figure 5.10: Prototype Overview [3](The 2D Image Plane Widget and the Histogram Widget), (Appercu du prototype, (Outil de vizualisation d' image en 2D plus l' outil de creation d' histogramme manuel))

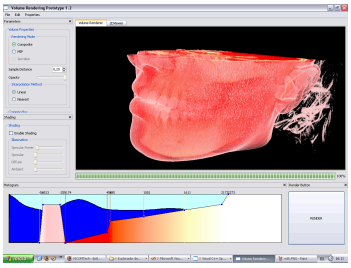

Figure 5.11: Prototype Overview [4] (Semi trensparent skin without shading), (Appercu du prototype, (Peau semi transparente sans effet d'ombrage))

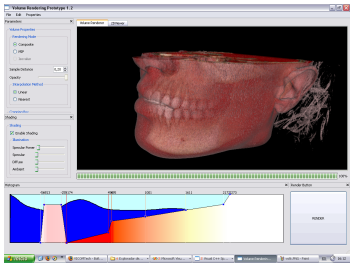

Figure 5.12: Prototype Overview [5] (Shaded semi trensparent skin), (Appercu du prototype, (Peau semi transparente avec effet d'ombrage))

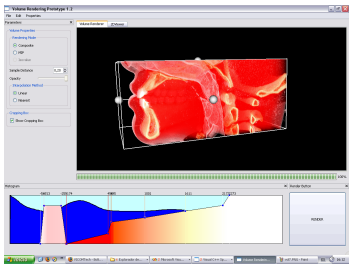

Figure 5.13: Prototype Overview [6] (Cropping box), (Appercu du prototype, (Cropping box))

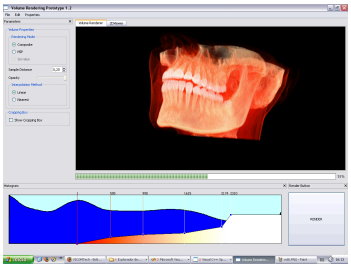

Figure 5.14: Prototype Overview [7] (Composite method without shading), (Appercu du prototype, (methode de rendu composite sans ombrage))

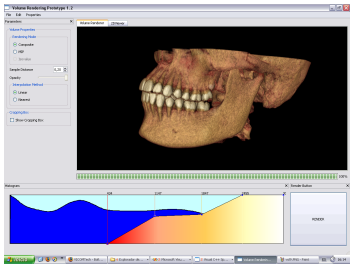

Figure 5.15: Prototype Overview [8] (Composite method with shading), (Appercu du prototype, (methode de rendu composite avec ombrage))

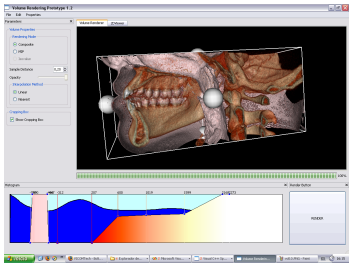

Figure 5.16: Prototype Overview [9]

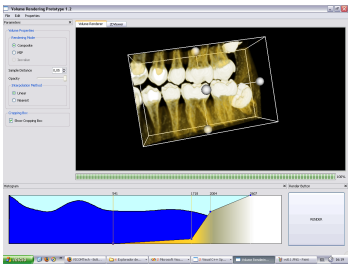

Figure 5.17: Prototype Overview [10]

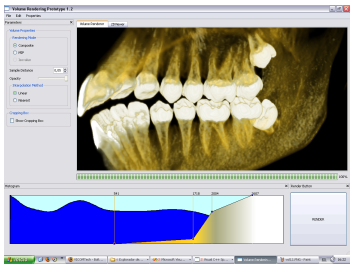

Figure 5.18: Prototype Overview [11]

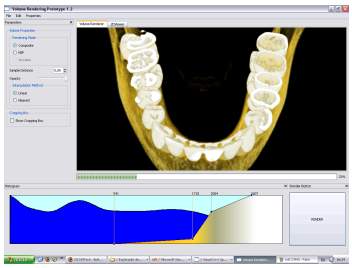

Figure 5.19: Prototype Overview [12]

Chapter 6

From Prototype Commercial Software [Two months]

6.1 Introduction

Software development is a complex task. There is not only one way and many methods exist to help both the persons who specify the needs, the software designers and the developers. From a prototype to a real "commercial" software, a large gap exists. When a developper creates a prototype, he wants to obtain a result as fast as possible. Strict code style, Memory management and Software Design are almost optional. On the contrary, developing a final application needs some knowledge but also some experience. For example, it tooked me about a month to develop the prototype without help and more than two months to implement the same features in the final application.

The next section has been divided in two parts, the first section explain one of the tasks I had to do in the final application, implementing the 3D management part. Of course my mentor Ivan helped me to implement it. It s not possible to detail the explanation, because the software belongs to a company and also because, during the two and half months I worked on the project. In the next paragraph the management part will be described followed by an example of a problem I ran into. The second part will be dedicated to a section I found particulary interesting, the thread support.

6.2 Managing the object to display on the screen

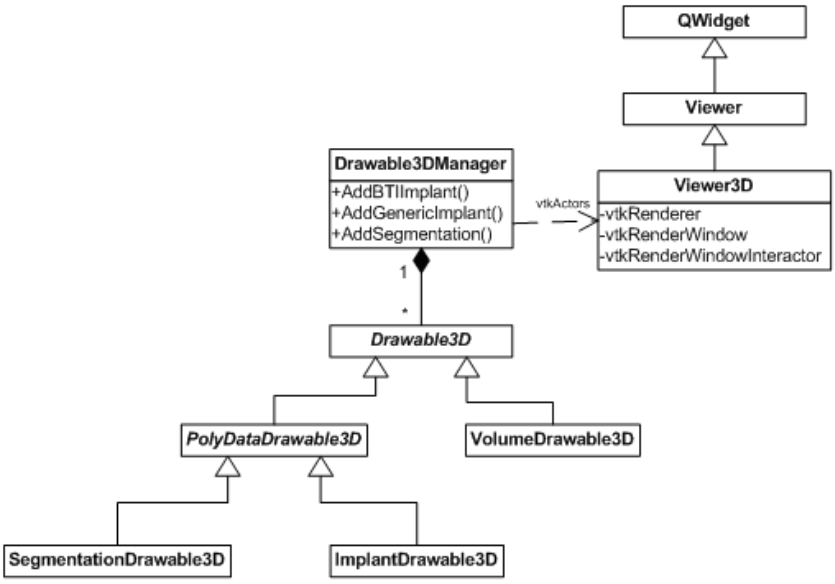

Figure 6.1: Software 3D design, (Design de La partie 3D du softaware IMP)

The (Figure - 6.1) presents the design of the drawable management part I was in charge of implementing. A drawable is something that can be renderered on the screen.

As it is shown, the Drawable3DManager contains a list of drawables smart pointers of different types: SegmentationDrawable3D, ImplantDrawable3D and VolumeDrawable3D.

The Drawable3D class is a pure virtual class and so no instance of this class can be defined. However C++ allows the creation of a list of Drawable3D smart pointers. Each of these smart pointer can point onto one of the different child class of Drawable3D.

The Viewer3D is a GUI class which allows to display some 3D objects on the screen. A 3D displayable object in vtk sense is an actor (vtkActor) The tcl-vtk example on the vtk example shows how to use such an object. Some additional information about the actors has been included in the Appendix.

In other words, the Drawable manager class contains the properties of the drawable, and the corresponding visual aspect of these objects is stored in the 3DViewer.

6.3 Implementation

Here will be presented one of the typical implementation issue I ran into and the way it was solved. When different people work on a same project, the work must be divided between them. One of the main difficulties encountered with software development, is how to connect the different parts the different people worked on.

The Implementation described in details here, deals with the picking process. Pick means to select something on the screen. This is part of user interaction and can be considered as an essential tool in

modern 3D applications (ex: Catia, SolidWorks ... etc).

In our case, the picking was used in order to enable the user to select an object or the background and change its properties (make it visible, change the opacity, the color, access to the control panel of this object... etc).

As shown in the previous section, the 3DViewer (from where the picking can be implemented) is totally independent from the DrawableManager. But selecting an object on the screen does not only mean selecting it 3D representation but also the Drawable which contains all the object properties. As a result, a way to go from picked vtkActor to a Drawable had to be found. The implementation guideline used during the software was to give priority to the communication between the class using the Qt Slot and Signal Mecanism [See the Appendix for more information about the Qt (signal and slot) mecanism).

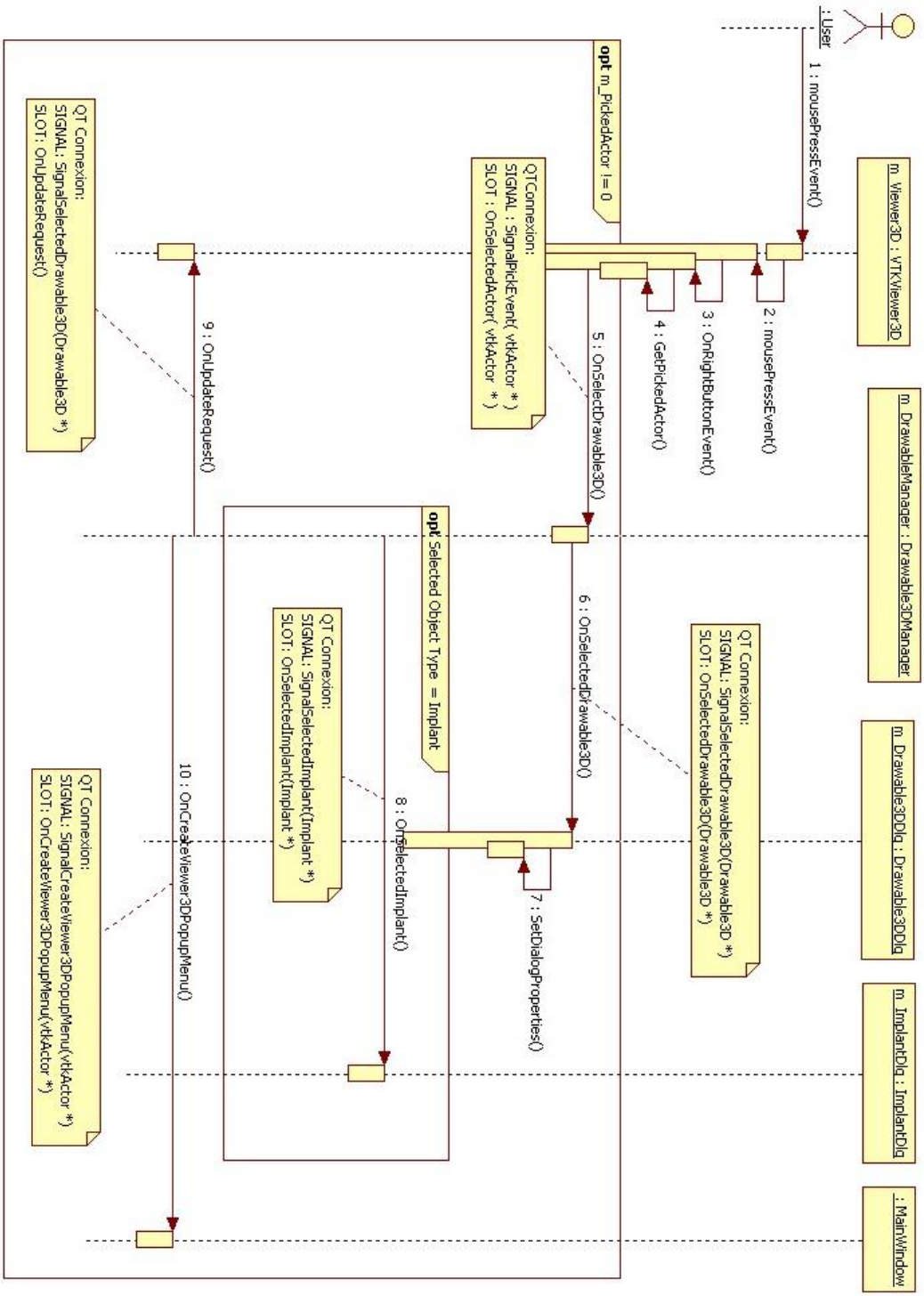

Figure 6.2: UML Picking diagram, (Diagram de séquence UML correspondant au picking)

The above diagram shows how the Picking process is computed. If a vtkActor is selected, the 3DViewer emits a signal to the DrawableManager specifying the selected vtkActor (a pointer is emited) Then the DrawableManager Search on its Drawable which Drawable has been selected (equality of the vtkActors).

Finally the DrawableManager calls the Drawable Dialog to show on the dialog the properties of the new selected drawable. Then some others actions are computed.

6.4 Implementing the 3D Viewer and its interactions

Most of the implemented tools in the first prototype had to be implemented in the final Application. I was in charge of implementing them. Some screen shots of the tools have already been printed in the prototype section so I will not go deeper into this subject here. Just to give an idea of the amout of work it represents, the number of lines in the header file of the 3D Viewer was about 500, and the .cxx file contained 1300 lines.

6.5 Application Optimisation

6.5.1 Introduction

Multithreading is treated as a black art by many programmers, and for good reason. Even for an experienced programmer, knowing when to multithread an application, where to multithread an application, and how to multithread an application can be difficult to determine. I will introduce here how the IMP application was threaded and why.

First of all, it is important to keep in mind that the size of the CT data we manipulated as the input of the application was high (between 50Mo and 100 Mo). The vizualisation pipeline that has been introduced in the last section can be considered as a list of sequencial operations. Each of these operations takes as input an important amount of data and transforms it into another representation. Each of these operations is CPU time consuming.

Threading these operations is necessary since it is not acceptable for a commercial software to freeze itself for a long time, disabling all the users controls. Moreover, the user wanting to launch a time consuming operation wants to know how much time he is expected to wait and how much time he still havs to wait before the expected result. The second reason which is more optional but which can be useful, is to be able to directly thread the pipeline internally in the case of a parallel one. An example will be explained later in this report.

6.5.2 Threading each pipeline step

The basic idea is to compute each step of the vizualisation Pipeline in separate thread from the main application one. In most cases, the pipeline is constituted by only a single dataflow, so that the n-ieme filter must wait for the output of the filter number n to be applied. When a threaded filter is finished, the hand returns to the main application which calls the next filter to be executed. Each threaded filter emits information about itself while it is running.

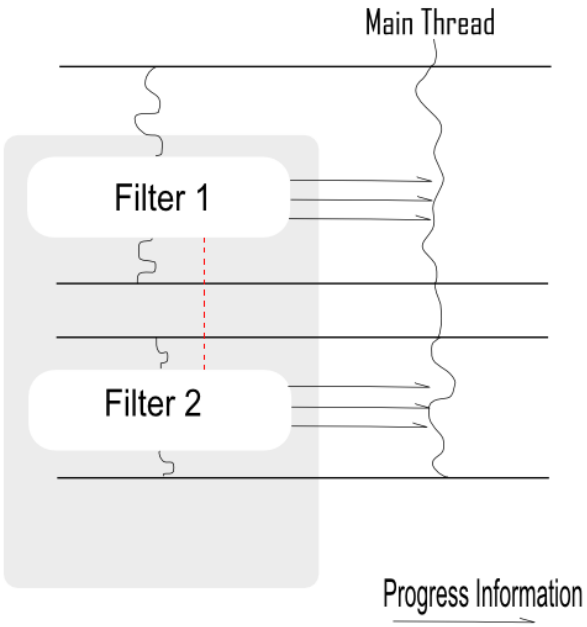

Figure 6.3: Model used to thread the pipeline, (Modele utilisé pour afin d'utliser l' option de thread)

The above picture shows an example of a threaded pipeline. The pipeline is constituted by two filters (filter 1 and 2). The main thread creates a new thread dedicated to the computation of the filter1, the thread that contains the filter1 sends to the main thread some status information about the calculation progress. Finally, when the filter1 has terminated, the hand returns to the main thread and the main thread calls the filter2 to be executed.

6.5.3 Implementation using the Qt Thread support

The Qt Library implements directly its own thread support as a class named QThread. A QThread represents a separate thread of control within the program; it shares data with all the other threads within the process but executes independently in the same way as a separate program on a multitasking operating system. Instead of starting in main(), QThreads begin executing in run().

Basically, in order to implement the threading support the idea is to execute each consuming processing time of the filter into the QThread run() method. In this way a base class for any threaded filter is created. The following diagram shows the way it was implemented.

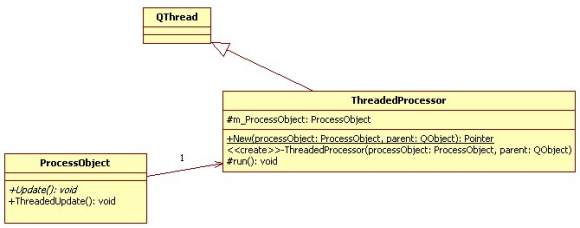

Figure 6.4: UML Diagram of the Threaded Process Object base class, (Diagramme UML de la classe mère possedant le support de Thread)

On the above UML image, only the main information has been kept. Deeper in the class one can find the progress support in order to watch the progress of the calculation.

6.5.4 C++ simplified implementation

As said above, the implementation of the progress state will not be explained here. The following lines of code show how the C++ code is internaly structured.

The ThreadedProcessor class

```
class ThreadedProcessor : public QThread
{
public:
  /** Constructor, the time conuming object must be set using the constructor */
  ThreadedProcessor(ProcessObject * processObject)
  {
    m_ProcessObject = processObject;
  }

protected:
  /** Threaded function, exectute the time consuming operation of the process object */
  void run()
  {
    m_ProcessObject->Update();
  }

protected:
  ProcessObject * m_ProcessObject;
}
```

The ProcessObject class

```
class ProcessObject
{
public:
  /** Ask the m_ThreadedProcessor to execute the Update function trough a thread */
  void ThreadedUpdate()
  {
    m_ThreadedProcessor = ThreadedProcessor::New( this );
    m_ThreadedProcessor->start();
  }
  /** Must be implemented */
  virtual void Update() = 0;
}
```

Implementation of a threaded Visualization pipeline filter

```
class Filter1 : public ProcessObject
{

  public:

    /** Set the inout of the filter */
    void SetInput(InputType * input);

    /** Get the output of the filter */
    OutputType * GetOutput();

    /** Calculate the output of the filter */
```

```
13      virtual void Update()
14      {
15        //calculation of the output using the input
16        //and the mITKFilter ...
17      }
18
19    protected:
20
21      itk :: filterType mITKFilter;
22  }
```

The way an application can be threaded really depends on the task the developer wants to thread, In the case of a vizualisation pipeline, the choice of threading each step of the pipeline can easily be understood. That allows for example the abortion of the process at each junction of the pipeline, and prevents the interface from freezing. However, even if this solution seems to be in adequation with the problem, the sequencial way the operation must be computed in a visual pipeline and requires the developper to be sure that one step is finished before starting the next one.

Chapter 7

Conclusion

From all points of view, the 5 months spent in VICOMTech were really rewarding. Technically, I improved my programming skills a lot. Supported by a highly skilled and really dynamic team, I realize now how much knowledge I gained during this training period.

The expertise I had to reach in order to optimize the surface rendering gave me an interesting approach into the investigation field. Even if the document I wrote will probably not be published, I have tried to write it as best as possible and it has been a very interesting task.

Unfortunatly I spent much my free time in Spain, in the Erasmus bubble, more than with the people of VICOMTech. As a result, I did not meet the Spanish culture as much as I had wanted.

But from a personal point of view, I improved my spanish a lot. I also improved my english by sharing my work and free time with people from every part of Europe. Indeed, the VICOMTech biomedical departement was composed by three natives, two German, one Tchek, and two French.

Appendix A

Example of an histogram filter using ITK

The following code presents a simplified version of an image histogram generator. This example first shows first how an ITK filter can be used and the aspect a encapsulate class can have.

In the first lines of code are defined the type of data which will be used. In this case the input data is an image of a short type and the output data is a std::vector which contains the values of the input image histogram.

A.0.5 .h file

```
1
2  class Histogram : public ProcessObject
3  {
4  public:
5
6    /** ITK Image type */
7    typedef short    ImagePixelType;
8    typedef itk::Image<ImagePixelType,3>                                    ImageType;
9    typedef itk::Image<ImagePixelType,3>::Pointer                           ImageTypePointer;
10
11   /** ITK Histogram typedefs */
12   typedef  itk::Statistics::ScalarImageToHistogramGenerator<ImageType>
         ImageToHistogramGeneratorType;
13   typedef  itk::Statistics::ScalarImageToHistogramGenerator<ImageType>::Pointer
         ImageToHistogramGeneratorTypePointer;
14
15   /** ITK Scalar range calculator typedefs */
16   typedef itk::MinimumMaximumImageCalculator<ImageType>                   MinMaxCalculatorType;
17   typedef itk::MinimumMaximumImageCalculator<ImageType>::Pointer
         MinMaxCalculatorTypePointer;
18
19 public:
20
21   void SetInput(ImageType * input)
22     { m_Image = input; }
23
24   /** return the output histogram */
25   std::vector<Double2Array> & GetOutput();
26
27   void Update();
28
29 protected:
```

```
30
31    //Histogram Outputs
32    std::vector<Double2Array> m_HistogramVector;
33    std::vector<Double2Array> m_LogHistogramVector;
34
35    /** input image */
36    ImageType * m_Image;
37
38  };
```

A.0.6 .cxx file

```
1   void Histogram::Update()
2   {
3     //...
4
5     ImageToHistogramGeneratorTypePointer imgToHst = ImageToHistogramGeneratorType::New();
6       imgToHst->SetInput(m_Image);
7
8     try
9     {
10      imgToHst->Compute ();
11    }
12    catch (itk::ExceptionObject& excpt)
13    {
14      std::cout << "ImageToHistogramGenerator Erreur " << std::endl << excpt.GetDescription() << std::
            endl;
15      return;
16    }
17
18
19    const ImageToHistogramGeneratorType::HistogramType * hst = imgToHst->GetOutput();
20
21    //...
22
23    for(int i =0; i < hst->GetSize()[0]; i++)
24    {
25      temp[0] = ...;
26      temp[1] = hst->GetFrequency(i)+1
27      m_HistogramVector.push_back(temp);
28    }
29
30    //...
31  }
```

Appendix B

The vtkActor object

vtkActor is used to represent an entity in a rendering scene. It inherits functions related to the actors position, and orientation from vtkProp. The actor also has scaling and maintains a reference to the defining geometry (i.e., the mapper), rendering properties, and possibly a texture map. vtkActor combines these instance variables into one 4x4 transformation matrix as follows: [x y z 1] = [x y z 1] Translate(-origin) Scale(scale) Rot(y) Rot(x) Rot (z) Trans(origin) Trans(position)

The following images shows how the actor is used in vtk:

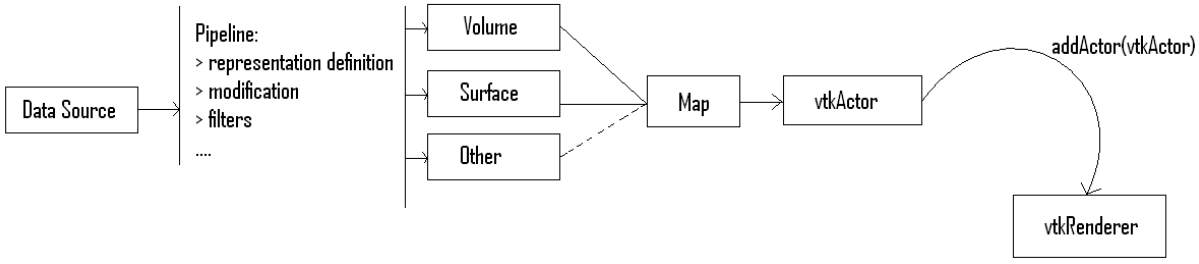

Figure B.1: Visualization process in vtk, vtkActor use, (le process de vizualisation à l' aide d un acteur)

Appendix C

QT Signal and Slot mecanism

This library offers the possibility to send and receive messages between different classes. The signal/slot mechanism is a central feature of Qt and is probably the part that differs most from other toolkits. This functionality can be very useful and easy to implement. Below a short example that shows how the SLOT/SIGNAL process works:

```
1  class A
2  {
3    Q_OBJECT
4    protected slots:
5      OnClassAEvent(char * msg)
6        { std::cout << "receive a message from class B: " << msg;}
7  }
8
9  class B
10 {
11   Q_OBJECT
12   signals:
13     void EmitMessage(char *);
14   public:
15     void SendMessage(char * msg)
16       { emit EmitMessage(msg);}
17 }
18
19 int main()
20 {
21   A a;
22   B b;
23   /* specify to Qt that "b" emit the EmitMessage(char *) signal and that "a" is able to recieve it with
         OnClassAEvent(char *)
24   QObject::connect(b,SIGNAL(EmitMessage(char *)),a,SLOT(OnClassAEvent(char *)));
25 }
```

Appendix D

The Segmentation process

In computer vision, segmentation refers to the process of partitioning a digital image into multiple regions (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image and make it more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images.

The result of image segmentation is a set of regions that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different in respect to the same characteristic(s).

In the case of jaws and teeth visualisation, the idea was to totally remove the air, the water and the tissues and only keep the bones, the teeth and the templates.

Figure D.1: An orginal CT Slice, (Coupe d'une image CT)

Figure D.2: The same image segmentated, (La meme image segmentée)

The main problem associated to the segmentation process is that this is a binary operation is a binary one. A pixel is set as in or out. So, the gradient of scalar values which exists in the case of an original image is replaced by a single in out boundary. This boundary can possibly deteriorate the visual result and the precision of the reconstructed model.

Appendix E

Improving Surface Rendering using ITK and VTK

Bibliography

[SAN04] D. SANTOS & BRODLIE, *LATEX : K. Gaining understanding of multivariate and multidimensional data through visualization, 2004*

[LIN01] LINDA G.SHAPIRO & GEORGE C. STOCKMAN, *Computer Vision, 2001*

[SCH04] W.SCHROEDER & K. MARTIN & B.LORENSEN, *The Visualization Toolkit, Third Edition, 2004*

[WIKI1] http://en.wikipedia.org/wiki/Visualization *Visualization (graphic)*

[VTK4] http://www.vtk.org/ *Visualization toolkit web site*

[ITK4] http://www.itk.org/ *National Library of Medicine Insight Segmentation and Registration Toolkit (ITK)*

[QT04] http://trolltech.com/products/qt *Cross-Platform Rich Client Development Framework web site*

[GOR98] Gordon Kindlmann 1 & James W. Durkin 2 *Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering, http://www.cs.utah.edu/ gk/papers/vv98/*

[LEV90] M. LEVOY *Efficient Ray Tracing of Volume Data*