Interactive Visualization of Biomedical Data

Daniela Gavidia Simonetti
B.S., Universidad Peruana de Ciencias Aplicadas, 2000
Supervisor: dr. Elena Zudilova
Mentor: dr. G.D. van Albada
Professor: prof. dr. P.M.A. Sloot

A thesis submitted to the Section Computational Science (SCS) at Universiteit van Amsterdam in partial fulfillment of the requirements for the degree Master of Science in Computational Science

July 2003

Acknowledgements

It's funny how life takes all these twists and turns and you can never really tell for sure what's going to happen next. A year and a half ago, I was in need of some intellectual challenge and eager to see the world. After searching high and low, I decided to apply for the MSc Computational Science programme. It didn't really sink in that I was leaving Peru until I had the plane ticket in my hands. I wouldn't have been able to make it that far without the support of the people at my home university, Universidad Peruana de Ciencias Aplicadas, who encouraged me to continue my education and always gave me great advice.

Of course, I want to thank my mother for supporting me and respecting my decision to travel so far away, even though I know she wanted me to stay. And my sister, for having such confidence in me and encouraging me to got for it.

A very big thank you to the people at Nuffic who let me apply for the fellowship by fax and skip all the embassy stuff. Otherwise, I would've missed the deadline. And, it goes without saying, that I thank them for awarding me a fellowship. Special thanks to the fellowships officers, in particular Lenneke and Rianne, for going to great lengths to ensure that the fellows enjoyed their stay in the Netherlands.

To the people of the Section Computational Science. I learned so much from you all. Special thanks to Elena for keeping me motivated and reviewing my thesis over and over, Denis for always having the answers to my technical dilemmas and Rob for being ever so helpful.

Last, but definitely not least, the friends I made in Amsterdam who made my stay so enjoyable. When I needed to take my mind off my studies and just enjoy life, I relied on you. Chris, Antony, Ting, I'm grateful for your friendship.

Contents

Abstract

Current 3D imaging systems have the capability to produce incredible amounts of high-resolution volumetric data. In order for this data to be analyzed, it has to be converted into visual representations. Specialists in medicine and biology rely on these visualizations to better understand the structures and relationships present in their subjects of study. Even though three-dimensional data is available, the traditional practice of analyzing the information using 2D slices is still predominant. This, of course, requires the expertise to be able to reconstruct 3D structures mentally.

Interactive visualization can bring to the forefront the information that might be obscured in the density of biomedical data sets, allowing the user to intuitively explore the data in real-time. More importantly, it has the potential of reducing the need for exploratory surgery and dissection by presenting clear, accurate 3D reconstructions of the objects of interest.

The work presented in this thesis intends to take advantage of the wealth of information in biomedical data sets by visualizing a 3D representation of the subject of study while at the same time using the familiar approach of visualizing 2D slices to explore the data set. Moreover, the flexibility of having a 2D slice that cuts through the data set in any arbitrary direction should give more insight into its content.

The need for accuracy as well as real-time response to the user's actions poses a challenge, given that the amount of information to be handled has a negative effect on performance. As a way to deal with this issue, heavy computations are being executed in parallel to relieve part of the burden on the client side of the application developed. The

purpose of this being that the resources on the client side should be used for providing interactive performance, while the computational power in the server side should be used to speed up the calculation of structures extracted from the data set. In this case, cutting planes.

Chapter 1

Introduction

1.1 Interactive Visualization

Visualization can be defined, in simple terms, as the transformation of data or information into pictures. This makes it easier for the human brain to process the information, as vision is the primary human sense.

"Visualization is a necessary tool to make sense of the flood of information in today's world of computers" [40]

This is true in every discipline that relies on computer power to generate, capture and process data. From market fluctuations to weather analysis to scientific simulations, such amount of data could not be handled by an ordinary person without reducing its complexity to aid its examination and understanding.

Through visualization, several goals can be achieved [10], such as:

- Explore available data at various levels of detail.
- Accomplish a greater sense of engagement with data.
- Give users a better understanding of data.
- Discover details, relations and patterns within the data through exploration

To reach these goals, the data - most likely comprised of a huge collection of numbers - needs to be converted to a representation that can be interpreted visually by the user, like geometric primitives and colors. As an example we can refer to the visualization of ocean salinity [2] where iso-surfaces are used to represent different levels of salinity and

the temperature of the waters is mapped into the color of the iso-surfaces. Without going into much detail and just to clarify this example, I should mention that an *iso-surface* is a surface composed of all the points within a volume of data that have the same value. They will be more extensively explained in Section 2.4.1.

In the field of medicine, the amount and complexity of the data available to medical staff is constantly increasing due to advancements in computer performance and storage capacity [10]. However, the availability of more data doesn't come hand-in-hand with an increase in human cognitive and perceptual abilities [5]. In fact, the shear volume of data can be overwhelming if not presented in an intuitive way. That is why visualization plays such an important role in the development of most kinds of medical systems.

Just like medicine, biological research [35] relies strongly on visualization for the study of structures and functional relationships. From cells and organelles to corals, biologic applications benefit from computer reconstruction and rendering of data captured by imaging techniques or generated by simulations [26].

By adding interactivity to a visualization system, the understanding of the relationships and patterns within the data can be improved. This means that the user is allowed to modify the presentation of the data, such as changing the coloring, transparency, viewpoint, etc..., with a high response time to his/her actions. In essence, an action by the user triggers a computation and the new presentation that is generated is displayed with minimal delay, so that the response to the users actions appears to be instantaneous. Due to the complexity of the data sets to be analyzed, interactive visualization brings all kinds of potential benefits in exploration by allowing the data to be displayed and manipulated intuitively and in real-time[1], with enough accuracy and speed to be a useful analysis tool.

1.2 Biomedical Data

In the context of this thesis, the phrase "biomedical data" is meant to be understood as referring to data collected from applying some kind of imaging technique to a subject of

[1] A more accurate way to put it would be *near real-time*, where the perception that the response to the user's actions is instantaneous is achieved by a response/repetition rate of 10 to 20 times per second [35].

biological or medical origin. Examples of this are the data sets produced by CT scanning a coral or taking an MRI scan of a patient in a hospital.

With the outstanding capabilities of new 3D and 4D imaging techniques and the increasing power of computers, the field of visualization of multidimensional data sets is finally living up to its promise of being the tool that would unlock the information hidden in biomedical data sets. Structures, properties and relationships would be revealed in an intuitive fashion through the use of image processing and visualization. As visualization becomes more prevalent, the need for intrusive surgeries or biopsies would be reduced, providing powerful new opportunities for medical diagnosis and treatment, as well as for biological investigations.

To illustrate this point, lets take a look at the role of medical imaging in the procedure of diagnosis and treatment of an illness. For example, in the vascular department of the *Medisch Spectrum Twente* hospital [12], all interventions are discussed in a meeting prior to implementation. Surgeons, radiologists and assistants gather together to look at all the data available for a patient. This includes CT (Computer Tomography) and MRA (Magnetic Resonance Angiography[2]) scan results. These medical imaging techniques produce three-dimensional data sets, but in practice a radiologists performs some post-processing in a computer and then prints out the resulting pictures. These are analyzed by the staff at the meeting in the traditional way, in 2D using a lightbox. It is up to the staff to mentally reconstruct the images in 3D using their expertise.

It is clear that the improvements in visualization have not catched up yet with the advancements in medical imaging, such as higher resolution of images and smaller distance between slides which yield more detailed information than ever before. The problem, though, is not the lack of techniques for extracting information from 3D data sets and visualizing them accordingly, but the shift that is necessary between the traditional practice of visualizing slides in two dimensions and doing so in 3D. In order for such shift to occur, applications that use three-dimensional data need to present it in a way that makes

[2]MR angiography is a study of the blood vessels using magnetic resonance imaging (MRI). It utilizes MRI technology to detect, diagnose and aid the treatment of heart disorders, stroke and blood vessel diseases.

exploration and analysis intuitive.

1.3 Goal of the project

The goal of this research project is to develop a visualization mechanism for the extraction of a two-dimensional slice from a data set by interacting with a three-dimensional representation of the object that was sampled to generate the data set.

The data sets considered for this kind of application have the potential of being extremely big. The advance in technology and imaging techniques suggests that these data sets could only increase in size, with higher resolution devices that offer more detailed information being in current development.

Having this in mind, the application developed for the purpose of this project has been implemented in a client-server model, where the bulk of that data is stored in a remote data repository accessed by the server. The client of the application interacts with a subsampled version of the original data set and can explore it interactively with a cutting plane. Once the desired cutting plane is found, the client is able to make a request to the server for a high-resolution version of this plane, generated using the original data set.

1.4 Chapter overview

This thesis is divided into five chapters. The following is a brief overview of the contents of each chapter:

Chapter 1 - Introduction - This chapter discusses the importance of visualization and how it applies to the fields of medicine and biology. It presents the goal of the research project and basic definitions and concepts needed for a good understanding of the following chapters.

Chapter 2 - Visualization for Medical Applications - Describes the imaging techniques used in medicine. One section of this chapter is dedicated image storage, focusing on the DICOM format, the predominant storage format for medical images. The size

of the files produced by medical imaging is discussed, which serves as motivation for developing our application in a client-server model.

Chapter 3 - Clipping engine - In this chapter, a thorough description of the implementation of the application is presented. The client-server architecture is explained in detail and subsections dedicated to the client and the server concentrate on the functionality of these two elements.

Chapter 4 - Results - Measurements of performance and an analysis of the project are presented. The focus of this chapter is to explore how interaction and user satisfaction is improved by the use of parallelism and sub-sampled data sets.

Chapter 5 - Conclusions - A final analysis of the development of this project is presented along with some remarks on the experience of the work done.

Chapter 2

Visualization for Medical Applications

Medical image visualization starts with the acquisition of data that represents the subject of study. This is referred to as *medical imaging* [35]. Today's imaging technology makes it possible to acquire digital samples of the subjects in three-dimensional space, i.e, in x, y and z dimensions. Once the data has been obtained, it has to be converted into a visual representation that can be interpreted and analyzed. Medical visualization applications and systems tackle this challenge by extracting the information available in medical data sets and creating an accurate presentation of such data through different visualization techniques.

This chapter covers medical visualization, from the acquisition of the data to the way it is stored and the techniques used to create accurate representations of the subjects of study.

2.1 Medical Imaging

2.1.1 Digital Imaging Techniques

The acquisition three-dimensional data from a subject can be done using different techniques. The choice of imaging technique is determined by the structure or anomaly that needs to be observed, given that some techniques are better suited than others for certain applications. When more complete information is necessary, different imaging techniques can be used to acquire images of the same object, complimenting each other. This is referred to as *Multi-modal Imaging* [35].

Through the use of specialized software, the results from different imaging techniques

can be merged into one representation. As an example, we can take the Erasmus Medisch Centrum, which uses the Radionics software to combine the results of CT and MR scans [13]. Using CT scans with a resolution of 512x512 pixels and MR scans with a resolution of 256x256 pixels, the software requires that the user indicate three points which are in the same position and uses this information to fuse the scan sets. In order to differentiate the data sets, CT data is then presented in red and MR in green.

Some of most commonly used imaging techniques are described in this section.

Computed Tomography (CT)

Computed tomography [31] is an x-ray based technique developed in the early 1970s, and now in widespread medical use. It revolutionized medical imaging and is considered to be the greatest advancement in radiology since x-rays. Three dimensional imaging is achieved by rotating an x-ray emitter around the patient, and measuring the intensity of transmitted rays from different angles. The result is a collection of 2D sections of the body that provide anatomical information on the positions of air, soft tissues, and bone.

CT scans are probably the most common source of three-dimensional data.

Magnetic Resonance Imaging (MRI)

In *Magnetic Resonance Imaging* (MRI) [39], also known as *Nuclear Magnetic Resonance Imaging* (NMR), the patient is placed inside a strong magnetic field usually generated by a large superconducting magnet. NMR is utilized to obtain images as a function of proton spin density and relaxation times. MRI is primarily used as a technique for producing anatomical images, but it can also give information on the physical-chemical state of tissues, flow diffusion and motion information.

Ultrasound

Ultrasound [39] is a real-time tomographic imaging technique. It produces real-time tomograms of the position of reflecting surfaces (internal organs and structures) and can also be used to produce real-time images of tissue and blood motion. High-frequency pulses of acoustic energy are emitted into the patient's body where they are reflected at

the boundaries between tissues of different impedance. By measuring the time delay and intensity of the reflected pulses, an image indicating tissue interfaces can be reconstructed.

Ultrasound is commonly used for imaging of the heart and blood vessels, as well as fetal development and gall bladder stones.

Positron Emission Tomography (PET)

PET [31] imaging begins with the injection of a metabolically active tracer molecule that carries with it a positron-emitting isotope. Within minutes, the isotope accumulates in an area of the body for which the molecule has an affinity. The radioactive nuclei then decay by positron emission. When the emitted positron collides with a free electron, high-energy gamma rays are produced which can be detected by an array of detectors in the scanner.

PET scanners are mostly used for brain imaging, in particular, diagnosis and localization of brain tumors and strokes, as well as monitoring blood flow changes associated with local brain function.

2.1.2 Window center and window width

This pair of parameters is used as a way of describing the *brightness* and *contrast* of the image. The common notation used for these parameters is C:W, where C is the numerical value of *window center* and W is the value of *window width*. Imaging techniques, such as X-ray/CT/PET scans, tend to generate consistently calibrated intensities. Therefore, a specific C:W pair can be used to visualize certain structures (e.g. 400:2000 might be good for visualising bone, while 50:350 might be a better choice for soft tissue). [36]

Reducing the window length increases the contrast, while displacing the window center alters the brightness of the resulting image.

2.2 Image Storage

There are several formats for image storage in current use. DICOM and Analyze are among the prevalent formats. Analyze[3] is actually an image processing program, written by the Biomedical Imaging Resource at the Mayo Foundation, which can read a variety of formats

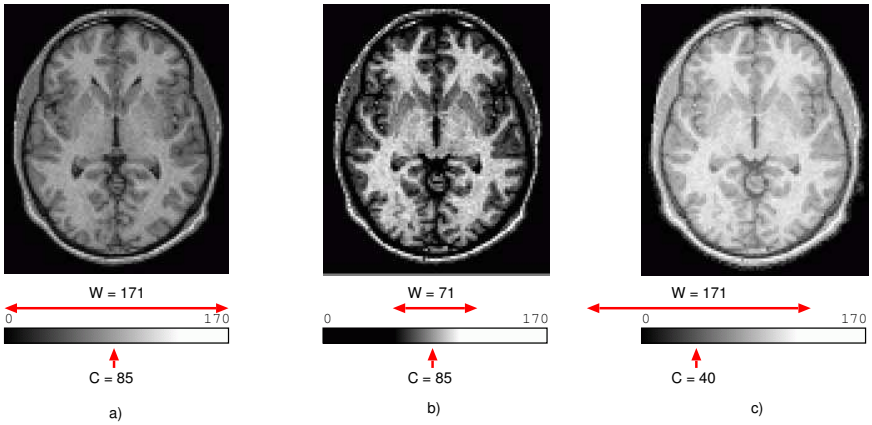

Figure 2.1: CT brain scan visualized with different pairs of C:W : a) 85:171, b) 85:71 and c) 40:171

and has its own formats as well. These are the Analyze 7.5 and Analyze AVW [8]. Image storage formats fall into two categories:

- **Standard** : These include ACR/NEMA 1.0 and 2.0 and its successor, DICOM 3.0, Interfile [18] (file format for the exchange of nuclear medicine image data), Papyrus [32],[34] (image file format based on ACR/NEMA 2.0, developed for the European project on telemedicine -TELEMED).

- **Propietary** : These include CT proprietary formats - like GE Genesis and Siemens Somatom - , MR proprietary formats - like GE Genesis, Siemens Magnetom and Philips Gyroscan - and other propietary formats not tied to a particular device, like the Analyze formats.

DICOM is the most common format for receiving scans. In fact, there are tools [11] available to convert most of the proprietary formats to the DICOM format. In the following section, the DICOM standard will be discussed.

2.2.1 The DICOM standard

With the introduction of CT scans and other digital imaging techniques in the 1970's, as well as the increasing use of computers in medical applications, the need for a standard method for transferring images and associated data between devices from different manufacturers became apparent. In 1983, the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) formed a joint committee to develop such a standard, with the following goals[29] in mind:

- Promote communication of digital image information, regardless of device manufacturer.
- Facilitate the development and expansion of picture archiving and communication systems (PACS) that can also interface with other systems of hospital information.
- Allow the creation of diagnostic information data bases that can be interrogated by a wide variety of devices distributed geographically.

Earlier versions of the standard were referred to as ACR-NEMA standards. The current version is the Digital Imaging and Communications in Medicine (DICOM) Version 3.0, which follows the ACR-NEMA standards version 1.0 and 2.0.

The DICOM standard is structured as a multi-part document consisting of 16 related, but independent parts. The complete documentation can be found at [29].

The DICOM format

Part 10 of the standard, *Media Storage and File Format for Data Interchange*, describes a file format for the distribution of images. Image files which are compliant with Part 10 of the DICOM standard are generally referred to as DICOM format files.

A DICOM file [36] contains both a header (which stores information about the patient's name, the type of scan, image dimensions, etc), and the image data (which can contain information in two dimensions (one slice) or three dimensions (a collection of slices forming a volume)). Image data is generally sampled using a resolution of 8 bits (256 levels) or

16-bits (65,535 levels), although some scanners can produce images with 12 and 32 bits per sample. RGB images are stored with three samples per pixel.

The DICOM image data can be compressed (encapsulated) to reduce the image size. Files can be compressed using lossy or lossless variants of the JPEG format, as well as a lossless Run-Length Encoding format (which is identical to the packed-bits compression found in some TIFF format images).

2.2.2 How big is big?

The volumetric data produced by the medical imaging techniques previously described can vary from relatively small (e.g. , 64 x 64 x 32) to huge (e.g., 2048 x 2048 x 1024) [21]. In most cases, the data consists of a single scalar value per pixel, but it is also possible to have three samples per pixel, as described in Section 2.2.1 (the DICOM format). Just to give an idea of the size of the data files, lets assume that there is one sample per pixel and each sample has a resolution of 8 bits. For the examples mentioned, the size of the data (not including the DICOM header) would go from 1Mb to more than 34 Gb.

2.3 Image Visualization

2.3.1 Image Viewers

The data sets produced by the medical imaging techniques described in Section 2.1 can not be analyzed by human staff in their original numerical representations. A conversion to a visual representation needs to take place. For this purpose, image viewers are available. Efforts go from simple viewers [37] to full-fledged image analysis systems [1], [25], [30], [23].

The majority of viewers open files in DICOM format and sometimes other popular formats too, like ACR/NEMA and Interfile. In addition to opening and showing the contents of the files, some viewers allow format conversion and image manipulation, like adjustment of brightness and contrast. More sophisticated applications, like 3D-Doctor [1] can perform 3D image reconstruction from a stack of 2D slices, producing volume and surface renderings. Specialized software, such as the Radionics XKnife RT system [47]

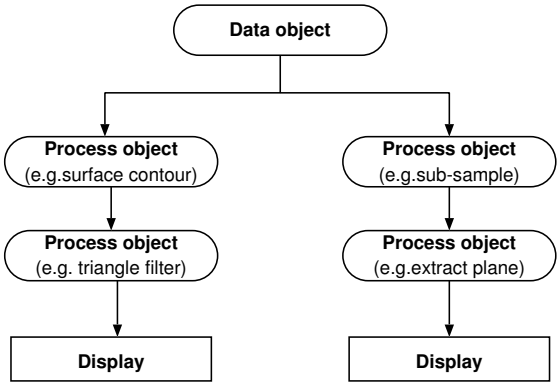

Figure 2.2: An example of a visualization pipeline.

used at the Erasmus Medical Centre, takes a step further and uses the data from CT and MR scans to generate a treatment plan (in this case, a radiation treatment plan).

Another category of viewers are the ones implemented using a client-server model, like SimpleDICOM [42] which allow viewing and exchanging files over a network. More ambitious efforts, like the Conquest DICOM server [45] from The Netherlands Cancer Institute, provide querying capabilities when communicating to the DICOM data base.

2.3.2 Visualization systems

Systems such as IBM Data Explorer [17], AVS Express [4] and the Visualization Toolkit (VTK) [46] have been designed to facilitate scientific visualization in general and can be used in medical applications. They provide numerous visualization and analysis algorithms for its users and all of them follow a data-flow model, where module inputs and outputs can be interconnected to create a *visualization pipeline*. In general, a pipeline consists of objects to represent data (data objects), objects to operate on data (process objects) and a direction of data flow (arrow connections between objects) [40]. An example of a visualization pipeline is presented in Figure 2.2.

IBM Data Explorer and AVS Express support threaded parallelism in a shared-memory environment. The Visualization Toolkit supports parallelism through the use of threads

as well as MPI[27][1] in a distributed system.

Unlike commercial products like IBM Data Explorer and AVS Express, VTK is a free, open-source project supported and extended by the contributions of a community of developers. It provides a wide variety of algorithms for visualization and image processing, allowing its users to render and explore 2D and 3D representations of their data.

2.3.3 DICOM to VTK format conversion

For this project, VTK has been chosen as the visualization library to use. In order to be able to process and visualize the DICOM data sets using VTK's algorithms, the data has to be read into a VTK object. DICOM is not among the formats[2] that can be read into VTK, so a conversion from DICOM to the VTK file format is necessary.

For the purpose of format conversion, a freeware tool called *dicom2* [15], developed by Sebastien Barre, has been used. It is a command line program which allows the user to convert images from the DICOM file format to various other formats, while optionally performing some rudimentary image processing tasks. *dicom2* can be used to convert medical images to raw format, which VTK is able to read. These can be 2D and 3D raw files, i.e. slices or volumes.

2.4 Visualization Techniques

Through the use of 3D imaging, it is possible to acquire digital samples of objects throughout 3-space (x, y and z dimensions). The collection of these sample values is known as a *scalar field*. There are several techniques that can be used to visualize static volumetric data. In this section, two common approaches for the visualization of biomedical data, surface rendering and volume rendering, are introduced.

[1] The Message-Passing Interface (MPI) is a standard for message-passing in a distributed memory environment. Message-passing is a paradigm widely used in high-performance computing to allow communication between different processes, making it possible to write programs that run efficiently on parallel computers with distributed memory.

[2] A `vtkDICOMImageReader` has been added to the nightly release of VTK. This suggests that the next stable release will support reading DICOM files directly into VTK. However, at the time of the development of this project, this capability was not available.

2.4.1 Surface Rendering

The collection of all points in a volume with a given scalar value is called a *level surface*, also known as *iso surface*. The scalar value that defines the iso-surface is called the *level* of the surface. A surface is generally constructed using polygons as primitives (although surfaces can also be represented by dots or vector nets). Standard computer graphics techniques, such as shading models (like Phong or Gouraud shading), can be applied to the polygon meshes and then the meshes can be rendered efficiently with the use of optimized hardware.

One of the most popular algorithms for iso-surface generation is the *marching cubes* [24] algorithm. In a three-dimensional scalar field, cells are defined as rectangular sub-regions with eight vertices. The algorithm assumes a linear variation in each direction within the cell. the idea is to march through the domain cell by cell and determine whether the surface passes through the cell. If it does, we need to determine the polygons that make up the surface. According to the set of vertices of the cell that are inside or outside the surface, a look-up table with the possible combinations of how the surface passes through the cell is used to determine the polygons to be used.

2.4.2 Volume Rendering

Volume rendering can be defined as the process of displaying scalar fields. The domain is a series of samples in 3-space, which translates to a voxel representation. Data acquired from imaging devices like CT and MRI scanners falls into this category. Each sample point in the domain represents an intensity averaged over a small sample volume. This sample value is constant over the voxel. Each sample value can be mapped to visual properties, like color and opacity, by applying a transfer function [9].

Volume rendering techniques based on ray-casting algorithms are widely used for the visualization of 3-D biomedical volume data. In this rendering process, the entire volume image data is used. The outcome visible in the resulting image is determined by the transfer function used to map the data to visual properties. The data that appears in the final image is obtained by firing a ray from a source point into the volume for each pixel

in the image, accumulating color and opacity as the ray intersects different cells. At each intersection, the scalar value is mapped to color and opacity via a transfer function and the values are accumulated until the opacity reaches unity or the ray exits the volume, whichever happens first.

Chapter 3

Clipping Engine

> "Accuracy is necessary when observing detailed quantitative behavior or comparing data. Interactivity is necessary to understand the overall structure of data or when looking for important features or other qualitative relationships" [22]

In an ideal case scenario, interactivity and accuracy would be possible simultaneously, but this is not often the case in real-world applications. For applications where heavy processing is involved and accuracy is a must, interactivity is hindered by the time required to do the computations. Therefore, a compromise has to be reached, where interaction is possible with an acceptable delay and the result presented to the user is detailed enough to allow a thorough understanding of the data that is being analyzed.

The struggle of finding a balance between interactive performance and accurate representation of the data, meaning taking advantage of the totality of the data available to generate images of the highest resolution possible, is at the heart of the development of this interactive visualization application. With this issue in mind, this chapter focuses on the design and implementation of the clipping engine. It starts with an explanation of the desired functionality of the application and what needs to be visualized. Then, the focus turns to the implementation in a client-server model, followed by individual sections dedicated to the client side and the server side.

3.1 Functionality

The goal of this project is to allow for the visualization of data resulting from medical imaging by interacting with a three dimensional representation of the data set. As it was

referred in section 3.2, this is achieved through the use of iso-surfaces as a means of giving the user an idea of the content of the data set and cutting planes to allow a more in-depth analysis of the characteristics of the observed data.

It is necessary to define what is understood as interactive visualization in the context of this project. Medical imaging provides a "snapshot" of the state of a subject at a given moment in time. Therefore, the resulting data is inherently static. Since this work focuses on a static dataset, the intention is to enhance the understanding of the structures within said data set through different rendering and color mapping techniques. This task is accomplished by allowing the user to interactively modify the viewpoint of the dataset, allowing for a thorough exploration, and giving him/her the ability to cut through it as desired, changing the position of the cutting plane as him/her sees fit. Interaction is made possible through the use of a graphical user interface (GUI) consisting of two visualization windows and sliders that control different parameters of interest. Section 3.4.3 describes the user interface in more detail.

The imaging techniques introduced in chapter 2 can produce enormous amounts of data, which is frequently stored in high-capacity storage facilities. In fact, one of the objectives for the introduction of the now widely popular DICOM standard was to promote the creation of such data repositories that could be accessed remotely by users. This leads to the natural choice of a client-server model for this application. Figure 3.1 shows a general view of the functionality desired for the client-server application being designed.

One of the main concerns for the user is real-time interaction. With this in mind, a subsampled data set is used for the interaction on the client side. This allows the user to manipulate the three dimensional representation of the data and choose the plane of interest in real-time. Once a selection has been made, the client communicates with the server to obtain the high-resolution version of the desired plane. The *sample rate*[1] of the data set used by the client can be modified by the user at any time, in case the default

[1] *Sample rate* is the frequency at which sample points are taken from the original data set. For example, for a sample rate of 2, one of every two points will be sampled, meaning that when traversing through the data set in certain direction, for each point sampled one point will be skipped. If the volumetric data set is sampled with a sample rate of r in the x, y and z directions, the size of the sub-sampled data set would be reduced by a factor of $1/r^3$.

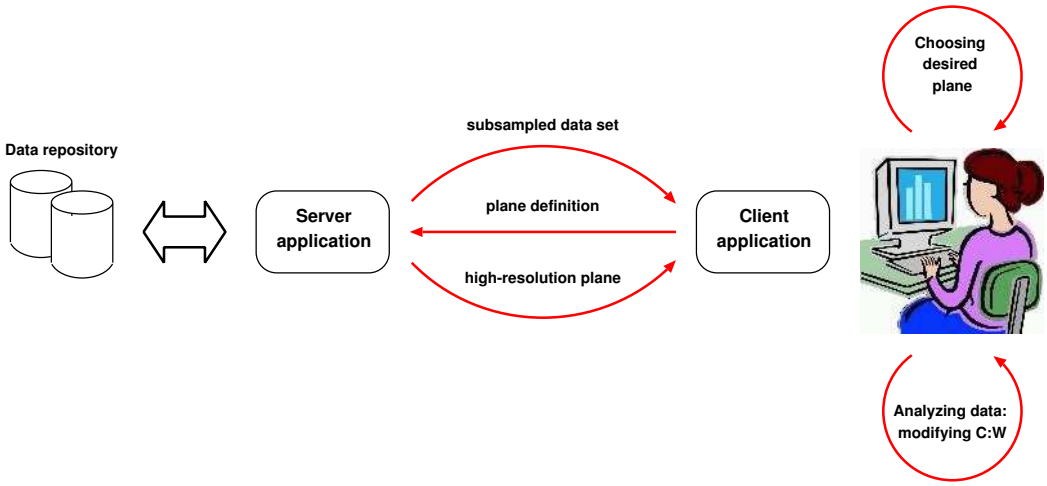

Figure 3.1: Retrieval and interactive visualization in a client-server model.

value is inappropriate for the resources on the client side.

Having the data sets stored in a remote location implies that they have to be sent over a network to the client. To make an approximation of the amount of time required for this operation, it is assumed that the request is made by the client by sending a message through the network and, as a response, the server sends the data set. The transmission time is made up of *latency* and *throughput*. The size of the message sent by the client to make the request is so small compared to the data set that it only contributes to the latency. From the moment the request is made by the client, the total transmission time can be defined by the following equation:

$$T_{trans} = 2 * latency + 8 * \frac{size\_of\_dataset}{bandwidth} \tag{3.1}$$

where *latency* is expressed in seconds, $size\_of\_dataset$ in MBytes and *bandwidth* in Mbps. The factor 8 is used to obtain the second term of the equation in seconds.

Just as an example, let's suppose that we want to send a 1 GB data set through a network with a bandwidth of 100 Mbps. Ignoring the latency, the transmission time would be 80 seconds.

Since the latency and bandwidth are inherent to the network, the only way to reduce

the transmission time is to reduce the amount of data sent through the network. One way of doing this is by sub-sampling the data set. Using the same sample rate r to sample in the x, y and z directions, the amount of data transmitted would decrease by a factor of $1/r^3$. The transmission time for a sub-sampled data set would then be:

$$T_{trans} = 2 * latency + 8 * \frac{size\_of\_dataset}{r^3 * bandwidth} \tag{3.2}$$

To illustrate the reduction in transmission time, let's assume that the 1 GB data set mentioned previously is sub-sampled with a sample rate of 2. The transmission time (discarding the latency) in this case would be 10 seconds, 1/8 of the original time required.

Another option that could be explored to reduce the transmission time could be sending the data sets in parallel. However, using parallel TCP is only beneficial when data has to be sent across very long distances, generally spanning several countries or between continents. To be more specific, when the round-trip time (RTT) is above 20 msec. In practice, data is sent in parallel between Amsterdam and Geneva, where the RTT is 25 msec. To do so across smaller distances would not bring any benefits.

For this project, sub-sampling is used to reduce the amount of data sent to the client, reducing the transmission time and the complexity of the data visualized on the client side. Whenever the user finds a cutting plane from the sub-sampled data set that is of interest, he/she can request the plane to be extracted from the original data set, receiving a high-resolution plane calculated using all the data on the server side. The transmission time for a plane could be approximated by:

$$T_{trans} = 2 * latency + \frac{resolution\_of\_plane * bits\_per\_sample}{10^6 * bandwidth} \tag{3.3}$$

where $resolution\_of\_plane$ is the number of pixels in the plane, e.g. 512x512.

For example, transferring a plane of 512x512 with 8 bits per sample would take roughly 0.02 seconds (just considering the throughput). It could still be argued that it might be better to transfer the entire data set first and calculate the cutting planes with the totality of the data on the client side, thus avoiding the requests of high-resolution planes to the server. It is necessary to consider that the transmission time of a plane is several orders of magnitude smaller than the time needed for the calculation of the plane itself. Taking

into account the fact that the computational resources on the server side, such as parallel computers, can provide significant speedups in the computation of the plane, it becomes apparent that doing this operation on the server side and sending the result to the client would be desirable.

3.2 Visualization Elements

> "Volume rendering is the preferred visualization algorithm for many types of 3D data sets, when it is not possible or not desirable to extract polygonized surfaces from the data." [21]

However, it is important to keep in mind that volume rendering implies rendering the totality of the data at the same time. For the amount of data that we are dealing with, this could prove to be prohibitive in terms rendering speed and slowing down the interaction considerably.

Another aspect of volume rendering that should be considered is the need for a transfer function to visualize the data. Finding a suitable transfer function to visualize the volumetric data is not a trivial problem.

> "There is a complex relationship between the transfer function and the resulting image. (...) the transfer function usually cannot be deduced from an idea of a suitable result image" [9]

Some imaging techniques, like CT, produce data that is standardized in the range of data values. Therefore, there are special subranges that always correspond to the same type of tissue. In these cases, once a useful transfer function has been found, it could be reused for all data sets of the same type. However, this is not the case for other imaging techniques, like MRI, where the data sets produced have different distributions depending on the patient, environment and certain machine parameters.

3.2.1 Isosurfaces

Since we do not wish to be restricted by the imaging technique used to generate the data, iso-surfaces are being used to represent the three-dimensional data. Isosurface rendering is less demanding because the amount of data presented at a given time is greatly diminished. Nonetheless, isosurfaces provide the user the possibility to explore the structures present in the data set. This is achieved by allowing the user to interactively set the iso-surface level.

3.2.2 Planes

The imaging techniques described in 2.1.1 produce a series of slides separated by a fixed distance. The collection of these slides make up a volumetric representation of the subject of study. The preferred method of analyzing data produced by medical imaging is through the use of two-dimensional slides, which are observed individually by qualified personnel.

In order to aid the analysis of the data, while conserving the traditional approach to which technicians are accustomed to, cutting planes are being used. However, the planes are not limited to being perpendicular to the axes. They can be oriented in any direction and translated to allow a thorough exploration of the data set.

3.3 Implementation

For the purpose of visualization and image processing, the Visualization Toolkit (VTK) [40] has been chosen. For this particular application, the filters for iso-surface extraction and sampling and clipping algorithms have been specially useful. Besides the visualization algorithms, support for parallelization and running remote processes has played an important part in the development of the application.

In order to allow the user to interactively modify parameters to explore the data, a GUI needed to be built. In the first stages of development, the Fast Light Toolkit [43] was used for this task. It's main selling point is being light-weight and small. In order to be used with VTK, separate threads for the visualization code and the user interface were necessary. Although, in the beginning this was not a problem, complications arose when

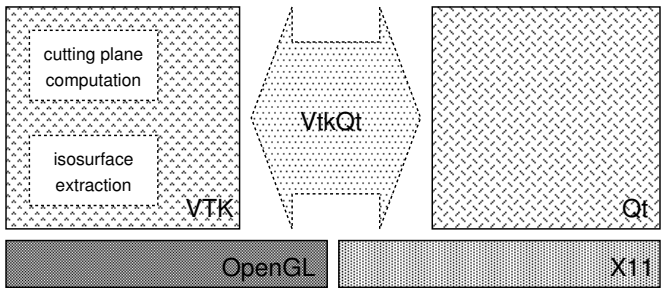

Figure 3.2: Software libraries used for implementation.

adding a second VTK window. Problems with rendering and updating the VTK windows could not be overcome. Since weight is not a big concern in the development of the user interface, it was decided to use another library for building user interfaces.

The final implementation was done using the QT GUI application framework [44]. Although not nearly as light-weight as FLTK, QT provides much better widgets and much more functionality. VTK is cleanly integrated into the QT interface through the use of the VtkQt library [41]. VtkQt is an extension of VTK. It is based on the VTK superclasses vtkImageViewer, vtkXOpenGLRenderWindow, vtkRenderWindowInteractor and Qt superclasses QWidget, QGLWidget therefore inherits all functionality of these classes. It allows for the insertion of VTK windows in the QT interface and transparent use of VTK classes alongside Qt, simplifying the development. Figure 3.2 shows the software architecture for this project.

3.3.1 Client-Server Architecture

There are several advantages to implementing the clipping engine using a client-server model:

- The computation of the cutting plane is expensive due to the need to find the points that make up the plane through interpolation. This can be quite time consuming. Assuming that better computational resources are available in the server, doing this operation in the server-side might yield better performance for the client.

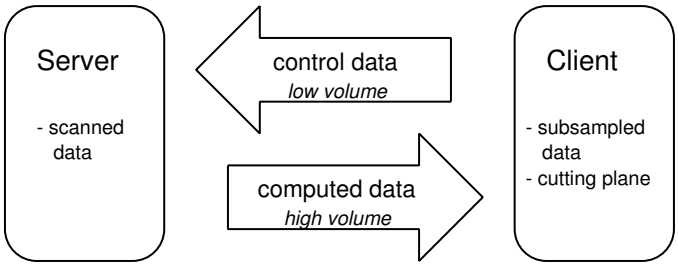

Figure 3.3: Client-server communication

- The data files can be contained in a central data repository. As explained in section 2.2.2, the size of these data files can go up to Gigabytes, making it inconvenient or not possible for the client to have them stored locally.
- Performance can be further improved by doing computations in parallel on the server side. It is unlikely that the client would have high-end computational resources, like a parallel super computer, at his/her disposition.

As a consequence of all the above, a client-server implementation allows the client to not burdened with the need to have to store enormous data files and have a fast processor to compute the planes. The main concern for the client would then be to have a fast enough graphics workstation for visualization and rendering.

3.3.2 Data Exchange

The flow of data between client and server occurs in an asynchronous manner, with the server sending subsets of the scanned data by request. The client and the server communicate through Remote Method Invocations (RMI).

Figure 3.3 illustrates the communication between client and server, pointing out the imbalance between the amount of data sent from client to server and from server to client. According to the flow direction, we define the data as:

Control data - data sent from the client to the server, comprised of the requests for

data made by the client, such as the definition of the desired cutting plane or the sample rate for a sub-sampled data set.

Computed data - data sent from the server to the client. In general, it is a subset of the scanned data sent to the client by request.

Control data is limited to a few bytes since it is comprised of requests for data, therefore the volume of data transmitted is low. On the other hand, computed data, being a subset of the scanned data, results in a significantly larger amount of information being sent (high volume). Due to the disparity in the volumes of data being transmitted, it is expected that, unless there is a high-bandwidth connection between client and server, more time would be required to complete the transmission of the computed data than to transmit the control data.

3.3.3 Operation and Communication

Figure 3.4 gives a general overview of the operation of the client-server application. The communication between client and server is triggered by two events: a request for a sub-sampled data set or a cutting plane. By default, a sub-sampled data set is requested when the application is launched. For this operation, a *sample rate* value is sent to the server, triggering the execution of a sampling routine and generation of the sub-sampled data set. In a similar manner, the client can request a cutting plane. In this case, the *plane definition* - a point in the plane and the normal of the plane - is sent to the server, triggering a routine for extracting the plane. The result is then made available to the client.

The communication between client and server is handled by the three classes: `vtkSocketController`, `vtkOutputPort` and `vtkInputPort`, as illustrated in Figure 3.4. The *control data* is sent from the client to the server through the `vtkSocketController`. After the necessary computations are performed on the server side, the resulting data sets (computed data) are sent back to the client through the `vtkOutputPort` and `vtkInputPort` objects. These two classes allow for the construction of a pipeline that starts on the server side and continues on the client side. The `OutputPort` is placed at the end of the pipeline

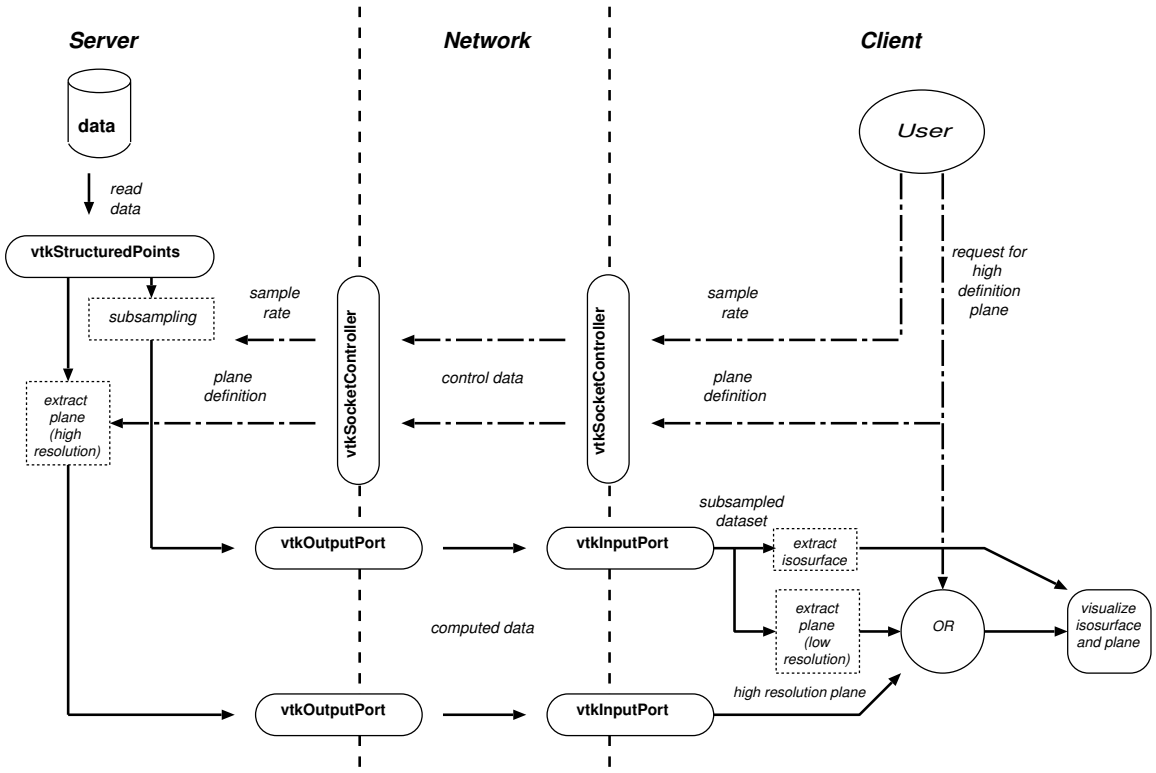

Figure 3.4: Pipeline of the operation of the client-server application. The communication between client and server is handled by `vtkSocketController`, `vtkOutputPort` and `vtkInputPort`

in the server and the `InputPort` at the beginning of the pipeline in the client. Together, they communicate all the pipeline protocol so that the fact the pipeline is running on different processes is transparent.

Several RMI methods are written in the server and registered with the `vtkSocketController`. The `vtkSocketController` in the client connects to the one in the server using the name of the server and a designated port number. Once the connection is made, the client application triggers the RMIs to communicate changes in the parameters controlled by the user, namely sample rate for the data set and definition of the cutting plane. The server then modifies these parameters in the server-side pipeline and makes the results available

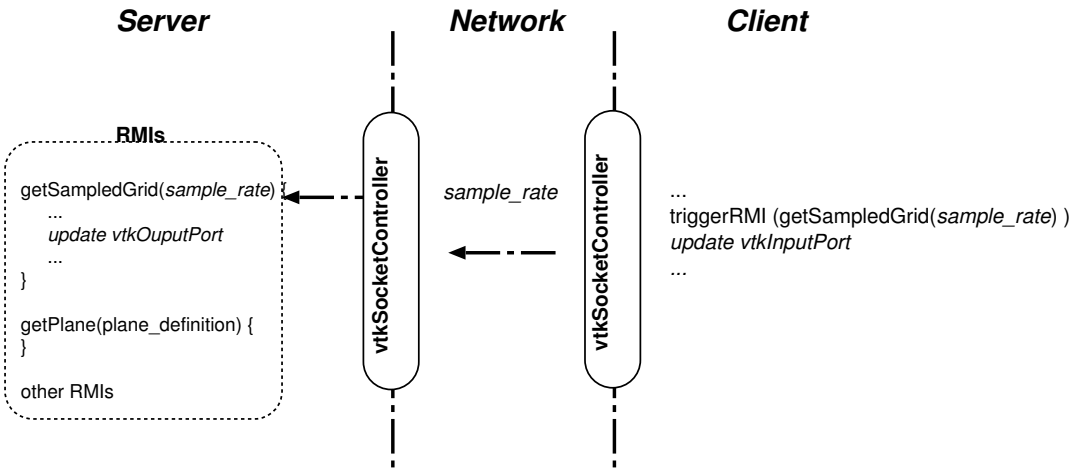

Figure 3.5: RMI being triggered from the client.

through the `vtkOutputPort` objects. After a RMI is made, the client updates the output of the corresponding `vtkInputPort` resulting in an update of the client-side pipeline. An example of the client triggering a RMI is shown on Figure 3.5.

With the sub-sampled data set, the client can generate locally iso-surfaces and low-resolution cutting planes. These are created according to the parameters set interactively by the user through the GUI. The iso-surface is rendered in a 3-D render window along with the low-resolution cutting plane. Only when a request is made to the server for a high resolution plane, is the high-resolution plane computed and rendered in the client side.

3.4 The Server

The server in the application is in charge of providing the client with the following:

Sampled data set - that can be used by the client for the three-dimensional representation of the data and interactive positioning of the desired plane.

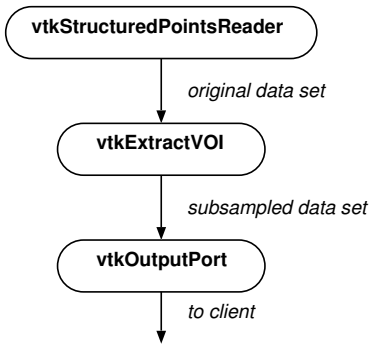

Figure 3.6: Pipeline of the sampling of the data set.

High-resolution plane - which is sent to the client as polydata, representing the geometric structure of the plane and point attribute values. Once received, it can be mapped to the desired color scale, allowing for the modification of brightness and contrast locally.

In the next sections, the implementation of these operations will be explained in detail.

3.4.1 Sampling of the Data

After the client sets the name of the data set to be analyzed through a RMI, the server loads this data into memory. The client can then request a sub-sampled version of the data set at any point by triggering a RMI with the chosen sample rate. This process is illustrated in 3.4. The data set can be sampled with different sampling rates in each direction (i, j, k). Figure 3.6 shows the pipeline for subsampling. The filter `vtkExtractVOI` is used to sample the data set with the sample rates defined by the user.

Figure 3.7 shows four snapshots of iso-surfaces of the same level generated from data sets obtained with different sample rates. From left to right, it becomes apparent that the reduced number of points in the data sets with higher sample rates produce less detailed results. Since a smaller amount of polygons is used in these iso-surfaces, faster

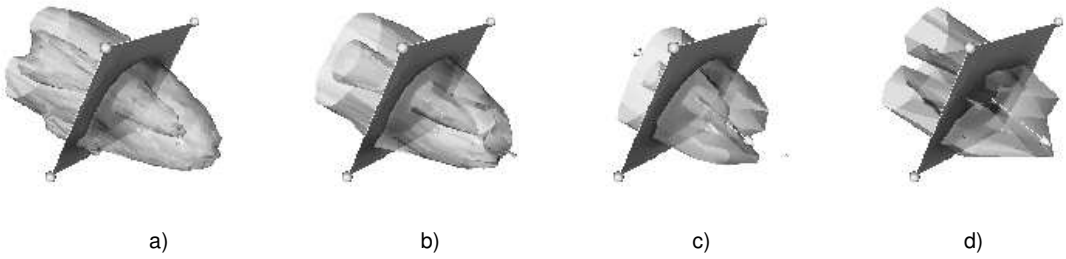

Figure 3.7: Iso-surface rendering for different sampling rates: a) sample rate = 1, b) sample rate = 2, c) sample rate = 4, d) sample rate = 5

rendering speeds are possible. So is the case with the plane that cuts through the iso-surface. Therefore, in the case where the original data sets are significantly big and the client doesn't have the resources to render a large amount of polygons at interactive speed, interactivity is aided by the use of a sub-sampled data set. Of course, information is lost by sampling the original data set. The motivation behind using a sub-sampled data set is only to improve the experience of the user in terms of interactive performance, allowing an overall exploration of the data set. For a more thorough study of a specific occurrence, the option of requesting a high-resolution plane is provided.

3.4.2 Generating the High-resolution plane

For the extraction of the arbitrary plane from the data set, the class `vtkCutter` was used. This class is a filter that cuts through a data set. It creates a polygonal surface corresponding to the implicit function F(x, y, z) = value(s) [20]. In this case, the implicit function was a plane function with the desired position and orientation. A pipeline for this process is presented in Figure 3.8. Cutting with an implicit function is time-consuming. Some measurements were made of the time taken to receive the high-resolution plane once the request is made by the client. Figure 3.9 shows the results. It can be seen that the time increases linearly with the size of the plane to be computed. The measurements

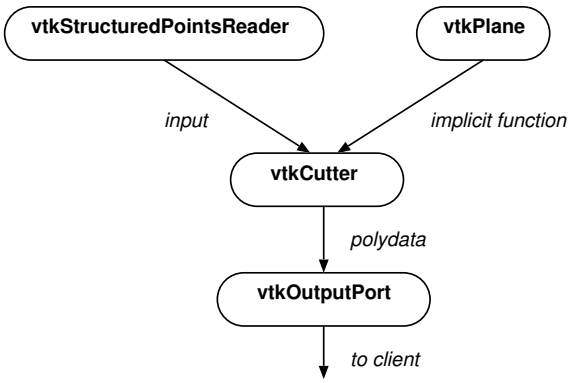

Figure 3.8: Pipeline of the extraction of the plane.

were made using data sets of different sizes. When asking for a plane that cuts through a data set of 512x512x53, the time required for the computation and rendering in the client was over a minute (the top point in the graph). This certainly goes against the goal of providing interactive performance. New measures should be taken then to speed up this process. Taking advantage of VTK's support for parallel computing, a parallel implementation of the plane computation is in order.

Data Parallelism

The Visualization Toolkit (vtk) supports task, pipeline and data parallelism [2]. Data parallelism is useful for processing extremely large data sets, not unlike the ones produced by medical imaging, as was addressed in section 2.2.2.

Data parallelism is achieved through the use of several parallel processes, where:

- an identical sequence of modules are run in each process.
- these data parallel modules process independent subsets of data
- the results of the last module in the sequence are usually merged to create a single result [2]

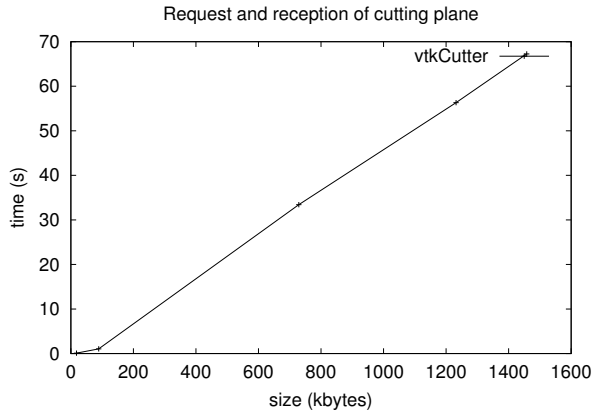

Figure 3.9: Time measurements for the interval between the request and reception of the high-resolution plane by the client.

This corresponds to the Single Program Multiple Data (SPMD) paradigm, where various parallel processes execute asynchronously one program on different pieces of data to achieve a faster execution of certain computation applied to a data set.

In order to reduce the time required to obtain the high-resolution plane after the request is made by the client, the calculation of the plane has been done in parallel. The implementation uses multiple processors that communicate through MPI, as shown on Figure 3.11. The processors execute the computation of the cutting plane on a different sub-volume of the original data set. The pipeline for each process, as seen in Figure 3.10 is similar to the one where the calculation was done only by one process (Figure 3.8), but it adds a step for selecting a different subset of data for each process, using `vtkExtractVOI`. The result generated by each process is a section of the cutting plane. These results are appended together by process 0 using a class called `vtkAppendPolyData` which has as inputs the sections of the plane calculated by each of the other parallel processes. The output of the `vtkAppendPolyData` filter is a single high-resolution plane (see Figure 3.11).

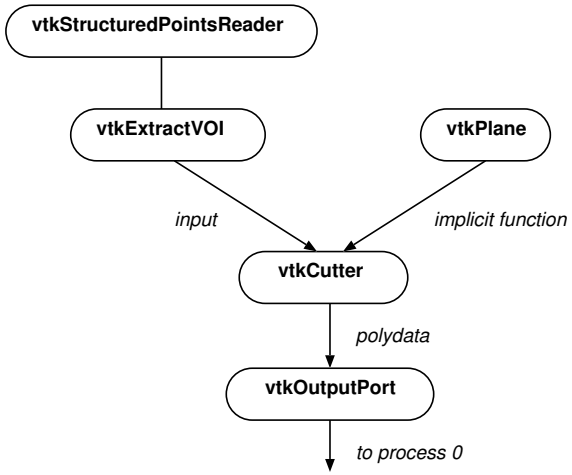

Figure 3.10: Pipeline of the cutting plane computation for each process that runs in parallel.

Data Decomposition

In order to achieve a reasonable *load balancing*[2] between the processors assigned to do the parallel computation of the plane, a suitable decomposition of the data must be applied. The normal of the plane has been chosen as the determining factor in the determination of the partitioning of the data domain. The domain decomposition is implemented in the following manner, where the total number of processors is *totalProcs* and the number of processors that calculate the plane in parallel is *nCalcProcs* = *totalProcs* - 1, because one processor is reserved for appending the results (see Figure 3.12):

- Determine the smallest component of the normal of the plane. Let's call it v.
- Divide the length of the domain in the direction of v into *nCalcProcs* segments of equal length. Let's call the length of each segment l
- Assign to each processor a rectangular block of data, where the length of the sides

[2]Load balancing is achieved when the workload is evenly distributed between processors, so that the time that processors are idle due to lack of work is minimized. When a processor is idle, processing cycles are lost resulting in the speedup being lower than its ideal value.

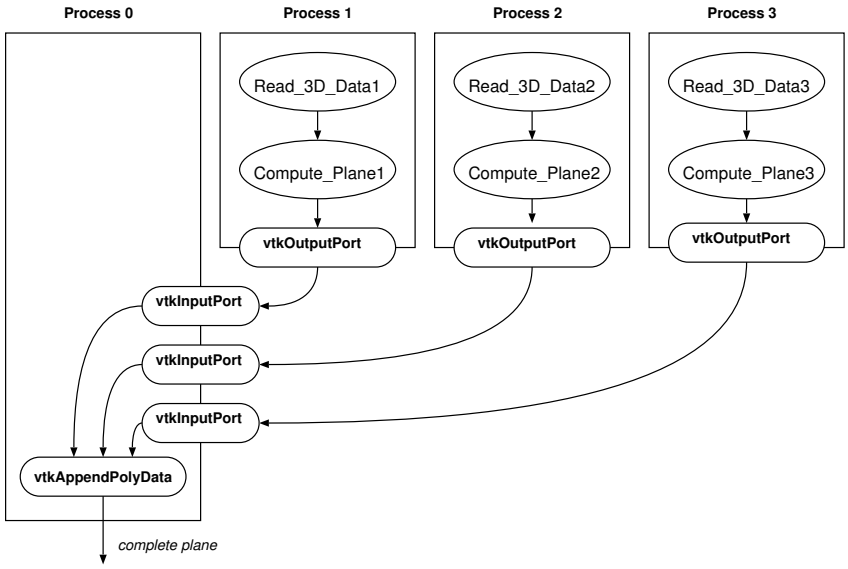

Figure 3.11: Data parallelism. Each process, except one reserved for appending the results, computes part of the cutting plane. In the figure, the computation is done using 3 processes. Process 0 appends the results from all the processes together.

of the block are the same as the original data set except in the direction of v. The length of that side is l.

The parallel implementation of the cutting plane computation was tested on the *Distributed ASCI Supercomputer 2* - DAS-2 [14]. This cluster uses the PBS [33][3] (Portable Batch System) to dynamically assign the nodes on which the computation will run. This makes it impossible to know in advance the name of the node on which process 0 would run, therefore there isn't a fixed node to which the client can connect to make its requests. To solve this problem, a stand-alone application has to run on the server. Let's call it the *request server* Process 0 of the MPI implementation connects to the *request server* and sends it the complete plane once the computation is finished. Since the *request server* has

[3]The Portable Batch System (PBS) is an extensible batch queuing and job management system for UNIX, originally developed for NASA. It operates on networked, multi-platform UNIX environments, including heterogeneous clusters of workstations, supercomputers, and massively parallel systems, providing a single interface to all computer resources.

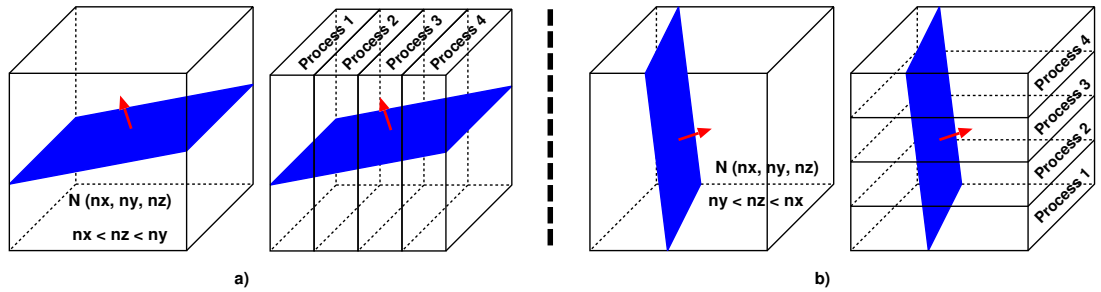

Figure 3.12: Two examples of data decomposition, with 4 processors. In a), the component of the normal in the x direction, nx, is the smallest of the components of the normal, so the partitioning is done in the direction of the x-axes. In b), the smallest component of the normal is in the direction of the y-axes.

a fixed location, any client can connect to it and make its requests. Therefore, the *request server* handles all the requests made by the client and in the case that a request for a high-resolution plane is received, the parallel plane computation module is executed.

> "in scientific visualization, three classes of interaction can be identified: interaction for configuration and control of the visualization pipeline, which is mainly done by means of graphical user interfaces, interaction for viewpoint control, and interaction with application data, also called semantic interaction" [16]

The GUI developed for this application allows the first two kinds of interaction described in the quote above. Control over the visualization elements, described in 3.2, is achieved through the use of sliders and direct interaction with the 3-D render window. The viewpoint can be modified by interacting with the 3-D render window, allowing for the examination of the objects from different angles.

3.4.3 GUI Elements

The GUI is made up of three elements: 3-D render window, 2-D render window and control panel. The arrangement of the elements is shown in Figure 3.13.

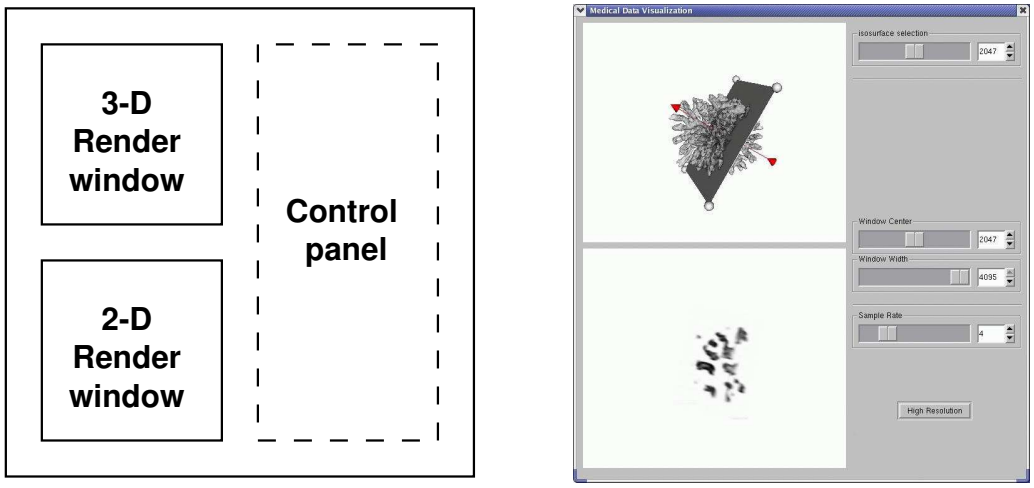

Figure 3.13: GUI mock-up showing the distribution of the elements (left) and snapshot of the GUI (right).

3-D Render Window

The functionality of this window is two-fold: it allows the visualization of data in three-dimensional space as well as direct interaction with the objects on the scene. The default interaction of the `vtkRenderWindow` is maintained, meaning that the user can rotate, translate and scale the objects by mouse interaction[4]. The 3-D render window is used to display two elements:

- Isosurface
- Low resolution cutting plane - calculated from the subsampled data set

The cutting plane can also be manipulated directly using the mouse. Its normal can be selected and set to the desired direction with the mouse and the plane itself can be dragged along the data set for exploration. This functionality is achieved with the

[4]The interaction is handled by the `vtkRenderWindowInteractor`, which forwards events to a `vtkInteractorStyle`. The default style is 'joystick', which means that holding down the mouse buttons generates a stream of events that cause continuous actions (e.g., rotate, translate, pan, zoom). It is possible to toggle between 'joystick' and 'trackball' style, which lets the user grab and move objects.

class `vtkPlaneWidget`. This class provides a plane that can be placed on the scene and manipulated by the user. Initially, the value of the center of the cutting plane is set to the center of the 3D data set and its normal to the z-axes. The exploration of the data set is achieved by implementing a callback function that executes whenever the `vtkPlaneWidget` changes position.

The visualization pipeline is set so that the plane used by the `vtkPlaneWidget` is used as the Input for a `vtkProbeFilter`. This filter samples data values at specified point locations. It requires two inputs: Input and Source. The Input geometric structure is passed through the filter and used to cut through the data set. The output of the `vtkProbeFilter` is the interpolated data values at the Input point positions [20]. In this case, we compute data values on a plane (plane specified as Input) from a volume (Source). In other words, we calculate the values of the 3D data set that fall into the plane. The callback function refreshes these values, which are mapped to grayscale according to the window center and window width parameters (which can be modified in the control panel) and displayed.

2-D Render Window

Unlike the 3-D render window, the 2-D render window doesn't allow direct interaction by the user. Its function is to display a front view of the cutting plane selected in the 3-D render window. This is accomplished by placing a camera focused on the center of the plane displayed in the 3-D render window and positioned at a constant distance from the plane. The view of that camera is displayed in the 2-D render window. To keep the correct viewpoint of the camera, its position has to be refreshed every time the position of the plane changes. This means that the camera position has to be reset in the callback function of the `vtkPlaneWidget`.

Control Panel

The widgets that allow the user to manipulate the parameters of the visualization are encapsulated in the control panel. The parameters that can be interactively modified are:

- Iso-surface level

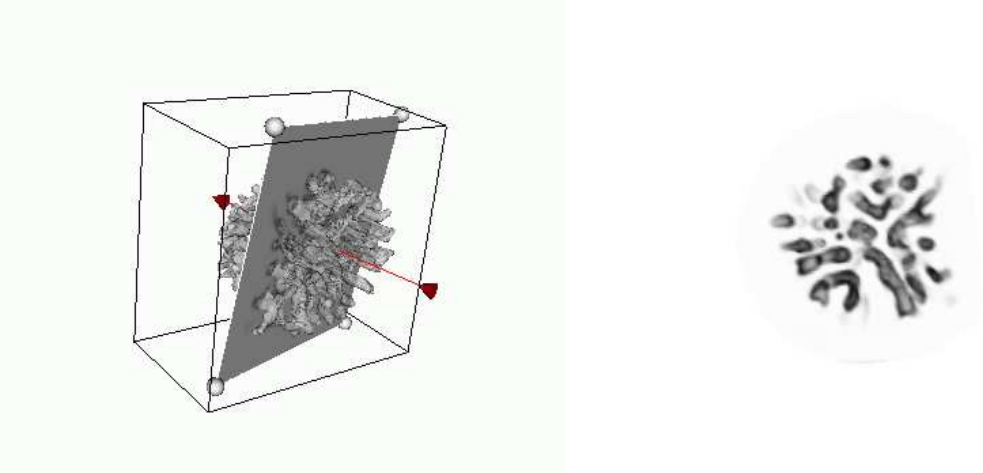

Figure 3.14: 2-D and 3-D render windows.

- Window center - which controls the brightness of the image
- Window width - which controls the contrast of the image
- Sample rate of the data set used for interactive manipulation in the client

The values of these parameters are modified through sliders. The range of possible values is determined according to the data set that is being analyzed. The minimum and maximum scalar values in the data set are obtained. With this information, the range of the window center is set to the range of values in the data set and its default value is set to the mid-point. The maximum value of the window width is set to the extent of the data set (maximum value - minimum value) and its default value is set this value, so that the default window covers all the scalar values in the data set. For the iso-surface level, the mid-point of the values in the data set is the default, with range of possible values going from the minimum value present in the data set to the maximum value.

3.4.4 Requests to the Server

As explained in 3.4, the client can request from the server sub-sampled data sets and high-resolution planes. This is achieved by using the widgets in the control panel. A slider allows the user to modify the sampling rate of the data set in use, thus making a request

to the server by triggering a RMI. A high-resolution plane can be obtained by clicking on a button, which results on a call to the appropriate RMI method in the server. In both cases, the requested poly data is received through a `vtkInputPort`. The sub-sampled grid and the plane each have a different `vtkInputPort` assigned for this purpose.

The update mechanism in VTK goes upstream, meaning that operations are not executed unless there is an explicit request for data. This means that although a RMI is triggered by the client, the data in the `vtkInputPort` will not be received until the output of the `vtkInputPort` is explicitly updated.

3.4.5 Local operations on the Client side

Once a sub-sampled data set is made available to the client, two operations can be performed: iso-surface extraction and generation of a low-resolution cutting plane. Section 3.4.3 explained how these operations are triggered through the GUI. Now, the implementation of these operations will be detailed.

Isosurface generation

The iso-surface generation is accomplished with the use of a `vtkContourFilter`. This filter takes a volume as input and generates one (or more) iso-surfaces as output using the Marching Cubes algorithm explained is Section 2.4.1. The result is a surface represented by a polygon mesh. In order to simplify the rendering of the surface, the output of the `vtkContourFilter` is passed through a `vtkTriangleFilter` to trangulate any non-triangular polygons. The result is passed through a `vtkStripper` to produce triangle strips and polylines, which are faster to render.

After the polydata is generated, it is mapped to a color and assigned a level of opacity to make it see-through, as to not obscure entirely the cutting plane that will be rendered with it in the 3-D render window. Figure 3.15 shows the pipeline for this operation on the left side.

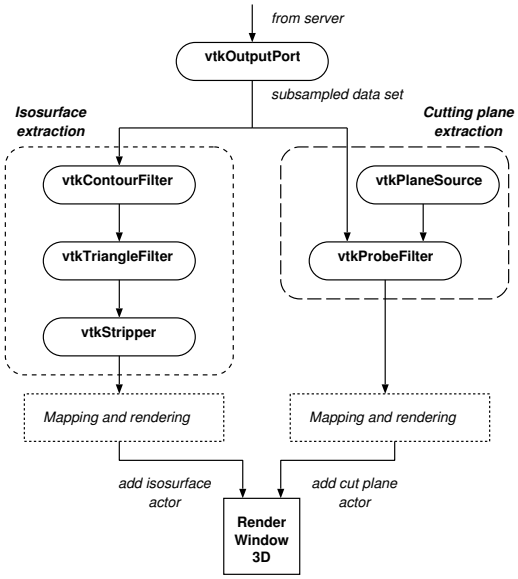

Figure 3.15: Pipeline for the processes applied to the sub-sampled data set received from the server. An isosurface and a cutting plane are extracted from the data set.

Low - resolution cutting plane generation

As explained in Section 3.4.3, a low-resolution cutting plane can be obtained from the sub-sampled data set using a `vtkProbeFilter`. The sampling of the data set is determined by the geometric structure used as Input, in this case a `vtkPlaneSource`. The `vtkPlaneSource` is a collection of quadrilaterals arranged to form a plane. The resolution of the `vtkPlaneSource`, meaning the number of quadrilaterals in which the plane is subdivided, is set according to the resolution of the data set. The `vtkProbeFilter` then samples the data set at the vertices of the quadrilaterals and the points in between are interpolated. Figure 3.15 shows the pipeline for this operation on the right side.

Chapter 4

Results

4.1 Parallel computation of the cutting plane

The parallel computation of the cutting plane on the server side has been implemented taking advantage of VTK's support for parallelism. Parallelism support is available in distributed-memory environments, via MPI, and in shared-memory environments, via pthreads or sprocs. The implementation was tested on the DAS-2 (Distributed ASCI Supercomputer 2). DAS-2 is a wide-area distributed cluster of 200 Dual Pentium-III nodes designed by the Advanced School for Computing and Imaging (ASCI). It is used for research on parallel and distributed computing by five Dutch universities (Vrije Universiteit, Leiden University, Universiteit van Amsterdam, Delft University of Technology and University of Utrecht). It consists of five clusters, located at the five universities (four clusters with 32 nodes and one with 72 nodes at Vrije Universiteit). The parallel implementation of the computation of the cutting plane was tested with up to 16 parallel processes.

Two approaches for data parallelism were tried:

- Using the data decomposition as explained in Section 3.4.2.
- Using VTK's support for data parallelism through parallel streaming.

VTK handles data parallelism through parallel streaming, where the data is partitioned into independent subsets using VTK's streaming data model [22]. In this context, streaming is defined as the ability of a sequence of modules to process an independent

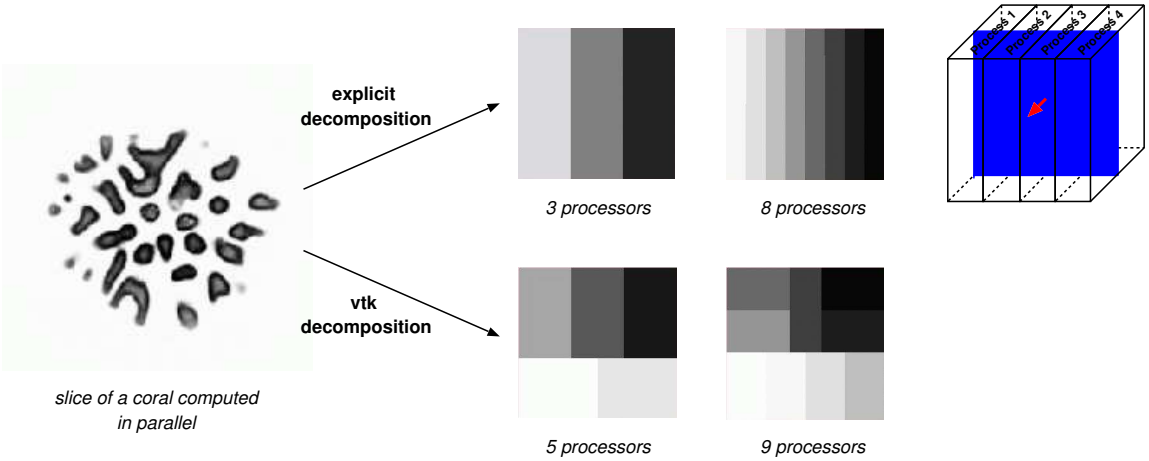

Figure 4.1: Domain decomposition using the explicit decomposition proposed in Section 3.4.2 and using VTK's support for parallelism. The sections of the slice computed by each processor have been rendered in a different shade of gray.

subset of data. Each parallel process executes a sequence of modules. The data partitioning starts when each process is asked to compute a portion of the entire output data of the last module in the sequence. At that point, for each of the parallel processes an update request is issued. In parallel, the update requests are propagated up the sequence of modules, determining for each module which portion of the input data is needed to compute the requested output of the module.

The `vtkAppendPolyData` filter (the one that appends the results of all the processes together to form the complete plane) implements this functionality. It's just a matter of activating the parallel streaming mode by calling `ParallelStreamingOn()`. Coding is simplified considerably since the developer doesn't need to explicitly assign a different subset of data for each process to work on. Instead, in parallel streaming mode the `vtkAppendPolyData` filter asks for a different piece of the plane polydata from each of its inputs (the outputs of the other processes). When the update request is propagated up the pipeline of each process, the request for a subset of polydata is translated into a

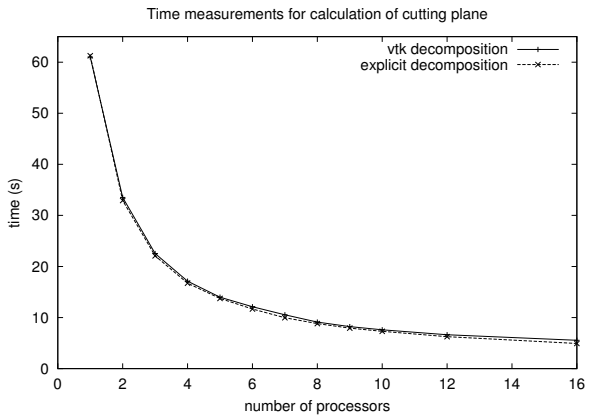

Figure 4.2: Time measurements for the calculation of a cutting plane of dimensions 512x512 with 16 bits per sample.

request for an independent subset of the input data set (i.e. the volumetric data).

Figure 4.1 shows the way the data domains are assigned to the parallel processes using the explicit decomposition detailed in Section 3.4.2 and using VTK's parallel streaming for the computation of a slice of a coral. The data set used for testing was a CT scan of a coral consisting of 53 slices with a resolution of 512x512 pixels. The plane to be calculated in parallel has a normal in the direction of the z axes, causing the volumetric data set to be divided as illustrated in the divided volume on the left side of Figure 4.1. On the other hand, the way VTK handles data parallelism produces domains of different topologies.

Figures 4.2 shows the time required to calculate the plane in parallel using an increasing number of parallel processes and Figure 4.3 presents the associated speedup. The proposed explicit decomposition yields slightly faster calculation times than the VTK decomposition using parallel streaming, despite more complex code needed to implement it. The difference is shown more clearly in the higher speedups obtained for the explicit decomposition.

It should be noted however, that the explicit decomposition works specifically for the calculation of a cutting plane and would not be appropriate if the application were extended to cut the data set with other surfaces. That would require a re-thinking of

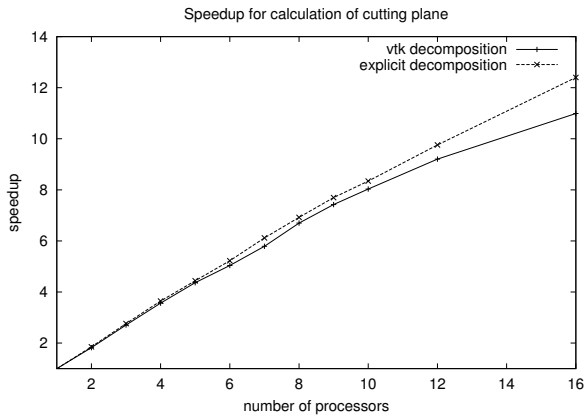

Figure 4.3: Speedup measured for the calculation of a cutting plane of dimensions 512x512 with 16 bits per sample.

the way the data decomposition is implemented. On the other hand, when using parallel streaming the data decomposition is not a concern and the system could be easily extended to use other surfaces to cut the data set.

Comparing the explicit decomposition versus the VTK decomposition using parallel streaming, the benefits in terms of improved performance are not very impressive. Although the explicit decomposition always yielded faster computation times, the margin by which it surpassed the other method was small, as observed in Figure 4.2. Considering the extra work required to implement data decomposition explicitly, the improvement it provides might not be considered significant.

4.2 Sub-sampled data set performance on the client

In order to improve the interaction speed in the client side, a sub-sampled version of the original data set is being used. The sample rate can be set by the user through the GUI. When interacting with the 3-D render window, the user perceives the interaction to be smooth when the response to his/her actions is (near) instantaneous. This is achieved with a response/repetition rate of 10 to 20 times per second [35].

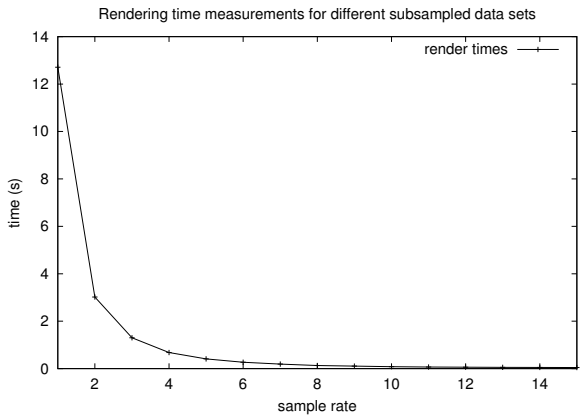

Figure 4.4: Measurements of the rendering times of the 3-D Render Window. The original data set (512x512x53) was sampled with different sample rates and these subsampled data sets were used to create an iso-surface and a cutting plane which were rendered in the 3-D Render Window.

To quantify the benefit of using a sub-sampled data set, measurements of the time required to re-render the 3-D render window have been made. The results are shown in Figure 4.4. The 3-D render window shows two elements, an iso-surface and a cutting plane extracted from the sub-sampled data set. When a transformation is applied to the elements (they are rotated, scaled, translated) or they are modified in some way (change in the iso-surface level, position of the plane, ...) it is necessary to render the window again. The complexity (number of polygons) of the data presented depends on the resolution of the data set, which has a direct effect on the time needed to render the scene. The rendering speed itself depends on the graphics hardware used [5], so the plots shown here should be interpreted in relative terms. Figure 4.4 shows the rate at which the rendering time decreases when the sub-sampled data sets are used. It is clear that even sampling every other point in the original data set (sample rate = 2) significantly decreases the rendering time.

The effect of sub-sampling in the improvement of the interaction is better observed in Figure 4.5, which shows the number of frames per second that are rendered for the different

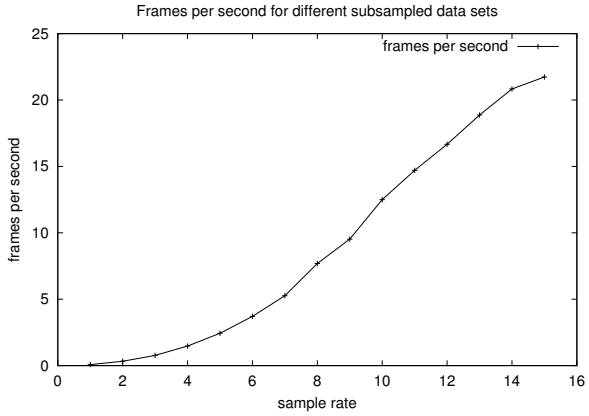

Figure 4.5: Frames rendered per second according to the sample rate applied to the original data set.

sub-sampled data sets. It is observed that only by sub-sampling it is possible to achieve an interactive response rate of at least 10 frames per second. Using the original data set introduces a considerable delay when interacting with the elements in the scene. Even when using the lower sample rates, the interaction less than optimal and the visualization is jerky.

Chapter 5

Conclusions

The goal of interactive visualization is to provide the ability to rapidly examine (and understand) data through visualization methods[19]. The work presented in this thesis strives to do just that, by using 3D visualization methods in an effort to make the visualization of biomedical data sets more intuitive, specially to the untrained eye that can not mentally reconstruct three-dimensional structures from a stack of 2D slices. At the same time, traditional 2D techniques are incorporated to accommodate the need for conventional analysis of 2D slices.

With the growing size of biomedical data sets, due to better resolution provided by the latest imaging techniques, centralized data bases with high storage capacity are becoming the norm. The application presented in this thesis has been developed assuming that the data sets are stored in a central location, making a client-server model the natural choice for implementation. The data sets are accessed by the server, to which the client connects to make requests for data. A sub-sampled version of the data set is used in the client side in order to achieve an acceptable visualization speed and make the experience of exploring the data set interactive, giving a near real-time response to the user's actions. However, this approach brings into question the trade-off between accuracy and interaction speed. Until high-performance graphics hardware is readily available on the average workstation, sub-sampling provides an efficient means to achieve acceptable interactive performance.

In order to deal with the delay introduced by the extraction of arbitrary 2D slices from a biomedical data set, the computation of these slices from the complete data set is provided by the server, where better computational resources are available, such as the parallel distributed supercomputer used for testing. By doing this operation in parallel,

it is possible to improve the interactive experience for the user. This was confirmed by calculating the speedup provided by the parallel computation, as was discussed in Chapter 4.

5.1 Discussion

For the exploration of the data in the client side, the main concern is interactivity, therefore a sub-sampled data set is used. However, the 2D slices extracted from a sub-sampled data set might not be detailed enough. Since the analysis of 2D slices in medicine is the standard way of visualization for diagnosis, it is of great importance to preserve the accuracy of the images. Therefore, it is desirable to use the totality of the data available for the extracted slice. This computation is done in the server, which has access to the entire data set. When the computation is done by a single processor, the performance of the application is crippled. As the size of the data sets increases, excessive amounts of time are required for the calculation of the cutting plane defeating the goal of interactivity. Doing the computations in parallel on the server side eases this problem, with performance tests showing the benefit.

Determination of an appropriate sample rate for the data set used in the client side is left to the user. Depending on the graphics hardware engine available in the client and the complexity of the data set, a higher or lower sample rate than the default value would be required for interactive performance. Finding this optimal sample rate falls beyond the scope of this project, but implementing an intelligent way of doing this would certainly prove beneficial. Moreover, in cases where the client is able to handle the entire data set without a noticeable effect on the interaction, it would be advisable to avoid sub-sampling, work with the entire data set in the client and reduce communication with the server.

5.2 Future Work

Biomedical imaging and visualization methods have the potential of being exceptionally powerful tools for clinical diagnosis and treatment, as well as offering the possibility of reducing the need for invasive clinical procedures. With the continued improvement in

3D visualization techniques, scientific exploration and biological investigations stand to benefit greatly from accurate representations of data where nature of the objects studied can be better understood through an intuitive exploratory process.

The appeal and acceptance of 3D visualization systems is not only linked with the advancement of the visualization techniques themselves, but comes hand-in-hand with the creation of improved interaction techniques and environments. It could be argued that the future of medical and biological investigations lies in fully immersive, real-time virtual environments. Virtual environments are defined as "realtime interactive graphics with three-dimensional models, when combined with display technology that gives the user immersion in the model world and direct manipulation." The key points here are interaction and immersion. The strength of virtual environments lies in that they allow an easy to understand presentation and more intuitive interaction with data. Virtual environments like the CAVE or the Distributed Real-time Interactive Virtual Environment (UvA-DRIVE) [6][1] could provide such functionality, although it could be argued that it is unlikely that their use outside a research environment would become widespread. A more portable and affordable solution could be the Personal Space Station [28], which permit a more natural interface with the virtual objects through direct handling.

Ongoing research at the section Computational Science of the UvA is focused on the development of a Virtual Radiology Explorer (VRE) [7] which allows the user to visualize and explore a patient's condition and experiment with treatments for vascular diseases. The work presented in this thesis can be thought as an element in such a system. The clipping engine is a desktop application, so integrating it into VRE would require a migration to a virtual environment implying a restructuring of the user interface.

The clipping engine has been used and tested in an academic environment. In real life, a variety of issues arise when sensitive information, such as medical data sets, is concerned. Remote access to patient data requires strong security and high-bandwidth is necessary for the volume of data to be transfered. Grid technology can provide this support

[1]The UvA DRIVE is a low-end VR system built with commodity hardware developed at Universiteit van Amsterdam in Asociation with SARA [38]. The system offers performance and features comparable to high-end systems, such as immersive projection, stereo vision and multi-modal interaction, at a relatively affordable cost.

and implementing the application as a grid service would seem not only convenient, but necessary.

Some more immediate improvements include direct reading of DICOM files and cutting of the data set with curved surfaces. The former can be easily accomplished when the VTK's reader for DICOM files, `vtkDICOMImageReader`, becomes available in a stable release. As it is now, the DICOM files have to be converted to raw data to be loaded by the application. The imaging data is conserved, but in the conversion process the header information included in the DICOM file is lost. By reading the DICOM file directly, this information could be preserved and the conversion step would be eliminated. Extending the application to include curved surfaces for cutting would enrich the possibilities for examining the data sets, such as vein irregularities.

Bibliography

[1] 3D-Doctor home page. On the web: `http://www.ablesw.com/3d-doctor/`. Able Software.

[2] Ahrens, J., Law, C., Schroeder, W., Martin, K., and Papka, M. A parallel approach for efficiently visualizing extremely large, time-varying datasets. Tech. Rep. LAUR-00-1620, Los Alamos National Laboratory, 2000.

[3] Analyze homepage. On the web: `http://www.mayo.edu/bir/Software/Analyze/Analyze.html`. Mayo Foundation for Medical Education and Research.

[4] AVS Express home page. On the web: `http://www.avs.com/software/soft_t/avsxps.html`. AVS Advanced Visual Systems.

[5] Belleman, R. *Interactive Exploration in Virtual Environments*. PhD thesis, Universiteit van Amsterdam, 2003.

[6] Belleman, R., and Shulakov, R. Distributed Real-time Interactive Virtual Environment (DRIVE) home page. On the web: `http://carol.wins.uva.nl/~robbel/DRIVE/`.

[7] Belleman, R., and Sloot, P. Simulated vascular reconstruction in a virtual operating theater. In *Computer Assisted Radiology and Surgery (Excerpta Medica, International Congress Series 1230)* (Berlin, Germany, June 2001), Elsevier Science B.V., pp. 938–944.

[8] Brett, M., and Rorden, C. The analyze data format. On the web: `http://www.mrc-cbu.cam.ac.uk/Imaging/analyze_fmt.html`. MRC Cognition and Brain Sciences Unit. University of Cambridge.

[9] Castro, S., Konig, A., Loffelmann, H., and Groller, E. Transfer function specification for the visualization of medical data. Tech. rep., Vienna Institute of Technology, March 1998. On the web: `http://www.cg.tuwien.ac.at/research/TR/98/TR-186-2-98-12Abstract.html`.

[10] Chittaro, L. Information visualization and its application to medicine. *Artificial Intelligence in Medicine 22*, 2 (2001).

[11] Clunie, D. A. Dicom3tools software. On the web: `http://www.dclunie.com/dicom3tools.html`.

[12] Cramer, H. Exploratory interview report 1: Medisch spectrum twente hospital. Universiteit van Amsterdam, 2003.

[13] Cramer, H. Report of meeting on radiotherapy planning at Erasmus Medisch Centrum. Universiteit van Amsterdam, 2003.

[14] Distributed ASCI Supercomputer 2 (DAS-2) home page. On the web: `http://www.cs.vu.nl/das2/`.

[15] dicom2 home page. On the web: `http://www.barre.nom.fr/medical/dicom2/`. Sebastien Barre.

[16] Hasse, H., Gobel, M., Astheimer, P., Karlsson, K., Schroder, F., Fruhauf, T., and Ziegler, R. How scientific visualization can benefit from virtual environments. *CWI Quarterly* (1994), 159–174.

[17] IBM Visualization Data Explorer home page. On the web: `http://www.research.ibm.com/dx/`. IBM Research.

[18] Interfile home page. On the web: `http://www.keston.com/Interfile/interfile.htm`. The Keston Group.

[19] Johnson, C., Parker, S., Hansen, C., Kindlmann, G., and Livnat, Y. Interactive simulation and visualization. *IEEE Computer* (December 1999).

[20] Kitware. Vtk Class Documentation. On the web: `http://www.vtk.org/doc/nightly/html/classes.html`.

[21] Koning, A. H. J. Applications of volume rendering in the CAVE. In *Paralleldatorcentrum Seventh Annual Conference Proceedings: Simulation and Visualization on the Grid* (December 1999).

[22] Law, C. C., Schroeder, W. J., Martin, K. M., and Temkin, J. A multithreaded streaming pipeline architecture for large structured data sets. In *Proceedings of Visualization '99* (October 1999), IEEE Computer Society Press.

[23] Ligier, Y., Ratib, O., Logean, M., and Girard, C. Osiris : A medical image manipulation system. *M.D. Computing Journal 11*, 4 (July-August 1994), 212–218.

[24] Lorensen, W. E., and E.Cline, H. Marching Cubes: A high resolution 3d surface construction algorithm. *ACM Computer Graphics 21*, 4 (1987), 163–169.

[25] Mazda home page. On the web: `http://www.eletel.p.lodz.pl/cost/software.html`. Technical University of Lodz.

[26] Merks, R. *Branching Growth in Stony Corals: A modelling approach*. PhD thesis, Universiteit van Amsterdam, 2003.

[27] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *The International Journal of Supercomputer Applications and High Performance Computing 8* (1994).

[28] Mulder, J. D., and van Liere, R. The personal space station: Bringing interaction within reach. In *Proceedings of VRIC 2002* (June 2002). On the web: `http://homepages.cwi.nl/~robertl/papers/2002/laval/paper.pdf`.

[29] NEMA. Digital imaging and communications in medicine (DICOM). Tech. rep., National Electrical Manufacturers Association, 2001. On the web: `http://medical.nema.org/dicom/2000.html`.

[30] Osiris home page. On the web: `http://www.expasy.ch/www/UIN/html1/projects/osiris/osiris.html`. Digital Imaging Unit. University Hospital of Geneva.

[31] Ozkurt, A. *Interactive Medical Volume Visualization for Surgical Operations*. PhD thesis, Dokuz Eylul Universitesi, Turkey, 2001.

[32] Papyrus home page. On the web: `http://www.expasy.ch/www/UIN/UIN.html`. Digital Imaging Unit of the University Hospitals of Geneva (Switzerland).

[33] Portable Batch System home page. On the web: `http://www.openpbs.com`. Altair Grid Technologies, LLC.

[34] Ratib, O., Hoehn, H., Girard, C., and Parisot, C. Papyrus 3.0 : Dicom compatible file format. *Med. Informatics 19(2)* (1994), 171–178.

[35] Robb, R. A. *Handbook of Medical Imaging: Processing and Analysis*. Isaac N. Bankman, Academic Press, 2000, ch. Three-Dimensional Visualization in Medicine and Biology, pp. 685–712.

[36] Rorden, C. The DICOM standard. On the web: `http://www.psychology.nottingham.ac.uk/ staff/cr1/dicom.html`.

[37] Rorden, C. ezDICOM home page. On the web: http://www.psychology.nottingham.ac.uk/staff/cr1/ezdicom.html.

[38] SARA home page. On the web: http://www.sara.nl.

[39] Schmidt, F. *Development of a Time-Resolved Optical Tomography System for Neonatal Brain Imaging.* PhD thesis, University College London, 1999. ch. An Overview over Existing Medical Imaging Techniques. pp. 35-44.

[40] Schroeder, W., Martin, K., and Lorensen, B. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics.* Prentice Hall, 1997.

[41] Shamonin, D. VtkQt. On the web: http://carol.wins.uva.nl/~dshamoni/VtkQt.

[42] SimpleDICOM home page. On the web: http://www.radiology.upmc.edu/Public/public_resources/software/index.html. UPMC Health Systems. University of Pittsburgh.

[43] Spitzak, B. Fast light toolkit homepage. On the web: http://www.fltk.org/.

[44] Trolltech. Qt homepage. On the web: http://www.trolltech.com/products/qt.

[45] van Herk, M., and Zijp, L. Conquest DICOM server. On the web: http://www.xs4all.nl/~ingenium/dicom.html. The Netherlands Cancer Institute.

[46] Vtk home page. On the web: http://www.vtk.org. Kitware.

[47] XKnife RT home page. On the web: http://www.radionics.com/products/rt/xkrt/. Radionics. Tyco Healthcare Group LP.