School of Computing and Mathematics

MSc Computer Aided Graphical Technology Applications

Franck Vidal

Constructing a GUI using 3D reconstruction for a radiographer's training tool

Submitted in partial requirements for the degree of MSc CAGTA

Sept 2002

Supervisor: Mr. S Keswani

Second Reader: Dr. J Warren

School of Computing and Mathematics

MSc Computer Aided Graphical Technology Applications

Franck Vidal

Constructing a GUI using 3D reconstruction for a radiographer's training tool

Submitted in partial requirements for the degree of MSc CAGTA

Sept 2002

Supervisor: Mr. S Keswani

Second Reader: Dr. J Warren

ACKNOWLEDGMENTS

I would like to thank Mr. Suresh Keswani of the School of Computing and Mathematics at the University of Teesside for his expertise. I would also like to thank the Senior Lecturer, Philip Cosson, of the School of Health at the University of Teesside, for suggesting this medical imaging project and supporting its development. I am grateful to Christian Girard of the Digital Imaging Unit of the University Hospital of Geneva for his assistance in my research and the use of his Papyrus toolkit. I would also like to thank the staff of the Newcastle General Hospital (NGH)for dealing with CT datasets and the radiography team of the South Cleveland hospital for showing how X-rays are taken. I would also like to thank Daniel Deprez for working on the companion project (the X-ray renderer) and his tutor, Julian Warren, for his participation in the project. I would also thanks Shona Davie for the proof-reading.

Abstract

Family Name	VIDAL
First Name(s)	FRANCK
E-mail Address	vidal.franck@voila.fr
Supervisor	KESWANI S
Second Reader	WARREN J
Course	MSc Computer-Aided Graphical Technology Application
Title	Constructing a GUI[1] using 3D reconstruction for a radiographer's training tool

Abstract

This project presents the GUI for a radiographers training tool. It is one of the two parts of a project idea proposed by Senior Lecturer, Philip Cosson, from the School of Health at the University of Teesside. His wish was to develop a program, which would allow students to train themselves to take X-Ray images without exposing patients to X-rays. It has been divided into two different projects, the GUI and the X-Ray rendering.

There are two parts to the GUI system, a 3D reconstruction and the setting of the radiography parameters via the GUI. Volumetric data, obtained by MR[2] or CT[3] scanners, is stored, slice by slice, in a DICOM[4] file, the medical imaging file standard. The Papyrus toolkit, developed at the University Hospital of Geneva, is used to read DICOMDIR files and DICOM files, which contain medical images. Before 3D reconstruction, information is extracted and a segmentation algorithm detects bones and skin on the different images of the dataset. From these data and using the marching cube algorithm, a 3D model is created.

The GUI lets users select different datasets. Users can set the position and the orientation of the 3D reconstructed objects. They are able to choose the corresponding cassette and to move the X-Ray source. The GUI communicates, via an agreed protocol, all settings of the scene to the other part of the project, the X-Ray renderer. The GUI project is written in C++ using the WIN32 API and OpenGL.

[1] Graphical User Interface

[2] Magnetic resonance

[3] Computed Tomography

4 Digital Imaging and Communications in Medicine

CONTENTS

INTRODUCTION

My project was proposed by the senior lecturer, Philip Cosson, of the School of Health & Social Care at the University of Teesside. He wished to have a training program for students in radiology. This product should allow students to simulate taking X-Rays without exposing patients to X-Ray beams. His initial project was divided into two different projects because of its size. This project is the creation of the GUI of the tool and the other project, which was produced by Daniel Deprez, was the X-Ray renderer.

The purpose of the interface is to allow the user to select the dataset for the training and then to set all radiography parameters, such as the patient position and the cassette, as in real conditions. For such training, users could select an actual dataset created by a medical scanner. The 3D objects are reconstructed from this dataset using the marching-cubes algorithm and they are used as part of the actual GUI. Image processing algorithms are applied to the obtained slices before reconstructing 3D objects. The rest of the GUI allows the user to set the position and the orientation of the 3D objects, the X-Ray source and other radiography parameters. This information is then sent to the companion project, which simulates the X-ray rendered images.

CHAPTER 1 - BACKGROUND OF THE PROJECT

The senior lecturer Philip Cosson, who specializes in medical imaging and diagnostic radiography, would like to have a X-Ray simulation program to allow students to train themselves to take X-Ray images without exposing patients to X-Ray beams. It is an important project in terms of working time. It was decided that this project would be divided into two different projects.

The X-ray simulation consists of a volumetric rendering. The simulator program has to consider physical parameters of X-ray and the position and orientation of each element that constitutes the 3D environment.

There are three main elements in the 3D scene; the patient, the X-ray source and the cassette. They have some properties, such as orientation and position. The GUI program sets up this information. It lets users select a dataset and set the scene as they want. The GUI calls the companion program, which displays the simulated final rendering. The GUI is divided into four parts; the first opens a dataset, the second is the reconstruction of two 3D objects, a skin model and a bones model. The third step is the setting of the 3D space and the last is the call of the rendering program. The simulator has been created by Daniel Deprez and the GUI by Franck Vidal, the author of this report.

Before starting the projects, a series of meetings were organized between everyone concerned in the project suggested by Philip Cosson. It was the starting point of the project. Philip's requirements were explained, the required and available data were defined. Some ideas were born, such as the use of a VR[5] device like a 3D mouse (a pen with a sphere) and the use of a grid for setting the position of the 3D objects.

The data available is of medical images of a phantom of a foot, obtained by a CT scanner at the Newcastle General Hospital (NGH). This hospital had scanned another object, a bucket with water, but a problem between different services of the hospital cancelled the obtaining of this dataset.

Philip Cosson had suggested a visit to the hospital to see how CT datasets are taken. This visit to the Newcastle hospital was cancelled, but another one was organized. This time it was to see how to perform real radiography. Cleveland South Hospital welcomed Daniel and

[5] Virtual Reality

me for a few hours to show us how to take X-Ray images. It was an opportunity to manipulate an actual X-Ray device. This visit was a useful experience because I could see what I was attempting to simulate and receive some ideas for the GUI. Using the same kind of buttons as the real device, would be an ideal method to simulate the process of taking X-Rays.

CHAPTER 2 - PREVIOUS WORKS

Before starting any implementation, other medical imaging programs were studied. My purpose was to discover what kind of results were obtained and what kind of interface was used. They were tested for positive and negative points. By critically appraising the programs' aspects, positive ideas may be used in the program.

At the beginning of the project, the phantom was scanned but the data was not burned on a CD because of problems between different services of the Newcastle hospital. At the beginning, the main activity was to research DICOM files as the loading of images from this kind of image files is necessary. As some Open Source DICOM files readers have already been implemented, these were searched for on the web.

By seeking and testing medical imaging programs, four different kinds of programs were defined; DICOM API, Medical image file viewers (with image processing), 3D reconstruction tools and X-Ray image renderers. As the purpose of the GUI is to send the companion program parameters to receive X-Ray images, X-Rays simulators were also sought.

2.1 DICOM API

As the purpose of the project is the creation of an interface, using 3D reconstruction, work was focalised on these two aspects, the interface and the reconstruction but images had to be read from standard medical image files. Using an existing library was preferred, rather than creating one, to allow more time to be spent on the main purpose of the project. Only two different libraries have been referenced; the Papyrus toolkit [6] and the Dicom3Tools [7]. As the program has to work under MS Windows, the Papyrus toolkit has been chosen because the other one only works in a Unix or a Mac environment.

Historically, the Papyrus toolkit is not based on part 10 of the DICOM version 3 standard, which defines DICOM files format. The Papyrus toolkit is anterior to version 3 of DICOM. It was created in 1990, when there was no medical image file format standard. The Papyrus toolkit defines a file format based on the data dictionary and data structure of the ACR/NEMA 2.0 communication standard (there is no concept of file format in this standard, the ARC/NEMA communication standard is the base of the DICOM standard). This file format responded to a need for an image storage and communication format. Later, the DICOM standard was born with its own file format. The Papyrus toolkit has evolved to have compatibility between Papyrus file and DICOM file. The toolkit can read both formats.

2.2 DICOM FILES VIEWERS

A good way to read images and to check the validity of the read data, was to read images from DICOM files and compare the result with other Medical image viewers, such as ezDicom [8], Osiris [9] and Rubo Medical Imaging [10]. These three programs were used during the implementation of the project. All of them have their advantages:

ezDicom is fast, it allows the user to change the contrast and the luminosity by clicking on the image. It shows the value of the pixel under the mouse pointer. This point is useful for seeing the result of a pixel value of a greyscale image encoded into a 16 bit colour depth (nowadays, computer monitors are not able to display more than 256 different greys).

The ***Osiris*** software lets users manipulate the dataset structure. The hierarchy of DICOMDIR files is shown. It is an interesting tool, because this program can be used as a Papyrus and DICOM file browser, which is very useful when images of a dataset have to be manipulated. As it has been created by the same team that created the Papyrus toolkit, the program is based on Papyrus. It is an example of what the Papyrus toolkit is able to do.

Rubo Medical Imaging shows and sorts all information contained in DICOM files. It is useful for checking particular record values when the new program reads DICOM files. It allows for the comparison of results obtained by this new program with results obtained by the Rubo Medical Imaging tool.

Another interesting program is ***Hipax*** [11]. It is a complete medical imaging manipulation tool. It can be used as a basic image viewer or complex image processing on a single computer; or on a larger network for PACS[6] solutions.

2.3 3D RECONSTRUCTION

Medicview 3D [12] is a program which can reconstruct 3D objects from medical images. There are two different kinds of rendering. The default 3D rendering uses a classical 3D object composed of triangles; this object seems to be reconstructed by a marching-cubes algorithm. The second method displays a 3D texture. The pre-calculation of the first method is longer than with the second, but, manipulation of the object is faster with the first method.

[6] picture archiving and communication systems

The most important weakness of this program is the difficulty of setting the orientation of the object in the space; rotations are not easy to manipulate.

2.4 X-RAY SIMULATORS

This project is not the only X-ray simulator that exists. Similar projects have been found.

The University of Umea , Sweden, has developed ***Virtual Radiography*** [13]. It is a tool to improve the learning experience of dental students in intra-oral radiography. Similar to this project, the 3D world is composed of the X-ray emitter, the patient and the X-ray detector. These can be moved. The manipulation of the objects can be done with the mouse, a spaceball or using a six degree- of-freedom tracker system.

The purpose of the second project, ***SimXray***, is to train students to take X-Rays without exposing patients and themselves to radiation. It is named ***SimXray*** [14]. It simulates an X-Ray image, using all parameters of radiography, as in a real-life situation. Users have to select examinations from a list and configure the X-Ray machine parameters. They can compare their results with a perfect X-Ray image.

The third project is not an X-Ray simulator, but it creates a virtual radiation laboratory [15] (X-Rays are radiations). Working with radiation is dangerous. People who work in such environments have to train themselves to detect all radiation sources, because the dangers are not always visible. This project is a VR trainer to teach users how to detect radiation sources. The user is placed in a laboratory setting with a certain number of sources in the room.

CHAPTER 3 - RESEARCH INTO ALGORITHMS FOR SURFACE RECONSTRUCTION

For creating my own medical imaging program, I have to know what algorithms are usually used for particular tasks in other medical programs.

One aspect of the project is the reconstruction of 3D objects from volumetric data. This version is formed by multiple 2D slices of computed tomography or magnetic resonance image data. Two possibilities were offered to display the object in 3D. The first uses voxels directly to perform a volumetric rendering as a 3D texture. The second was to produce an isosurface mesh[7]. The first method needs to store all slices in memory, as it is a 3D texture. The rendering is slow and uses a lot of memory. The second requires commonly used 3D objects represented by faces of three vertices.

One of the tested programs, ***MedicView 3D***, displays and manipulates a 3D volume (a head) in the space, using those two methods. One is very fast when compared to the other one. The method of isosurface extraction was implemented because the manipulation of such 3D objects can be achieved quickly in real time.

3.1 MARCHING CUBE [1]

This is an algorithm, which appeared in an article, in SIGGRAPH, written by W. E. Lorensen and H. E. Cline in 1987. The algorithm processes the 3D data in scan-line order and calculates triangle vertices using linear interpolation.

Marching cube locates the surface in a logical cube created from four pixels of a slice and four pixels of the adjacent slice (Figure 1). The algorithm determines how the surface, stored in the 3D data, intersects this cube, then it moves (or marches) to the next cube. Commonly, a byte is used to store the vertices' cube description. The convention used is to put a one to a cube vertex, if the data value at this vertex exceeds or equals the isovalue of the surface that is being constructed. Such vertices are on or inside the surface. Other cube vertices get a zero, because they are outside the surface. Cube edges are intersected by the surface when there are vertices inside and outside the surface.

[7] collection of triangles in 3D, joined along common edges and vertices

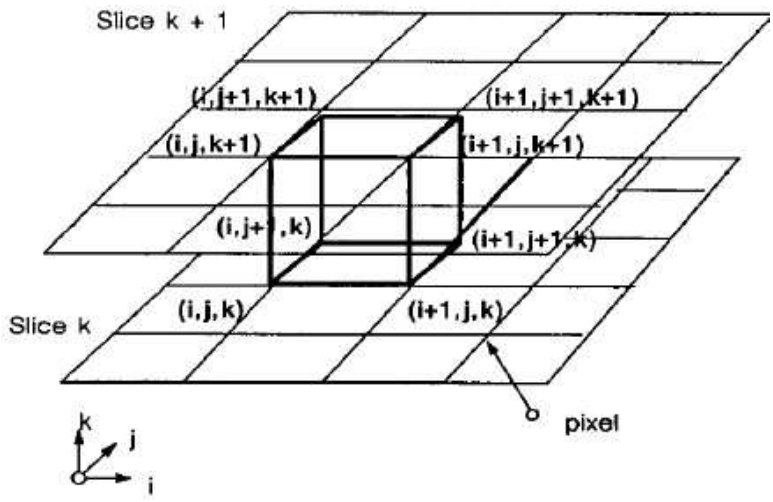


Figure 1. Marching Cube.

(Picture from the Lorensen and H. E. Cline's article in SIGGRAPH 87)

A cube has eight vertices. Those vertices can have two states, one or zero. There are only 8^2 = 256 different cubes. A lookup table is created by listing all different cases. It contains the edges intersected for each case. Two different symmetries reduce the problem to 14 patterns (Figure 2). Symmetry of states reduces to 128 possibilities. The second symmetry is done by rotation and reduces to 14 different possibilities. For each possibility, there is only one possible tessellation. Per cube, there can be 0 to 4 triangles.

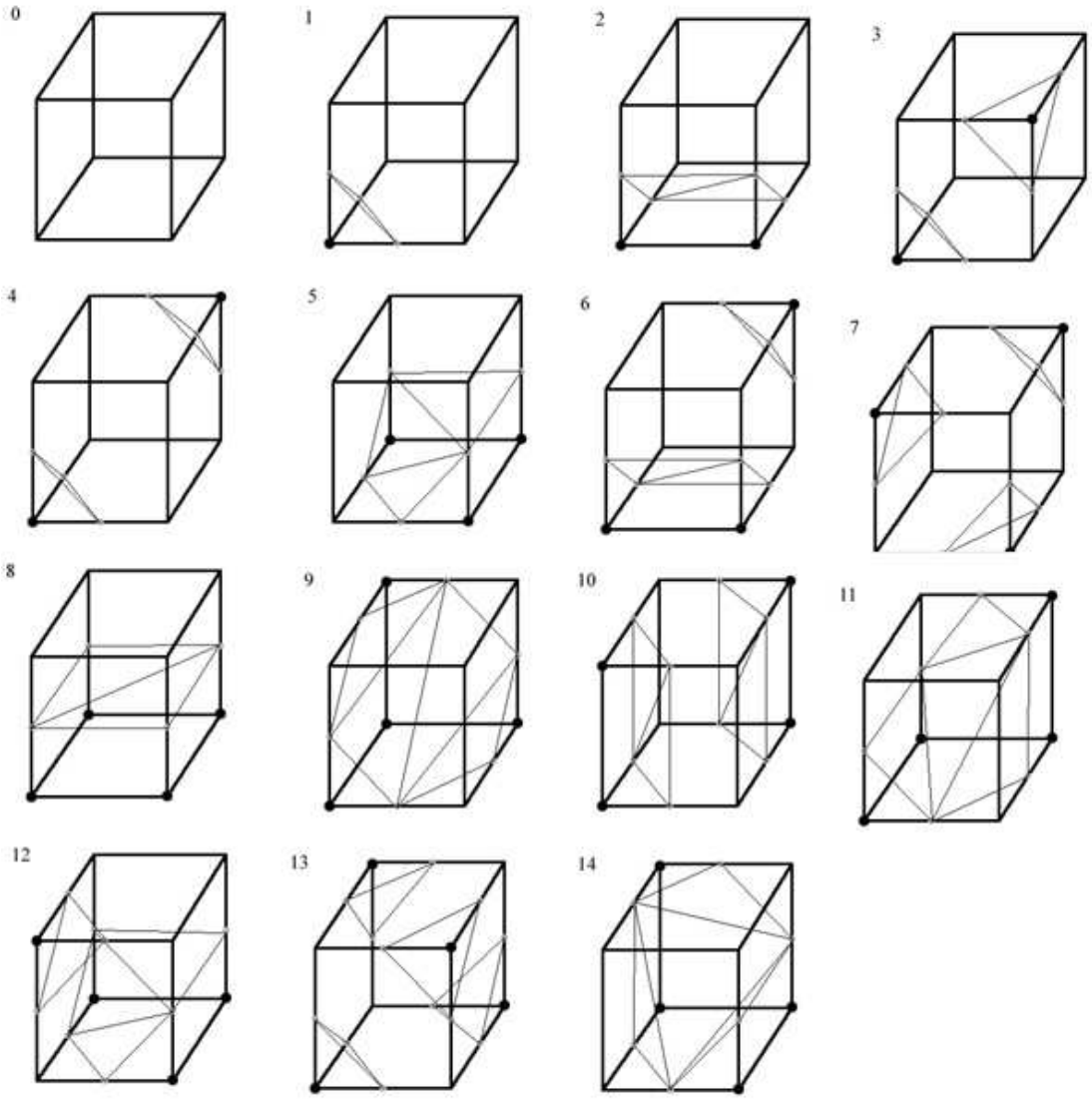


Figure 2. Base Patterns.

An index, based on the vertices' states, is created for each case (Figure 3). The index contains one bit per vertex. As there are eight vertices, which can have two states, just a byte is required to stock the description of cube vertices. The created cube is used as a pointer to an edge table, which indicates all edge intersections.

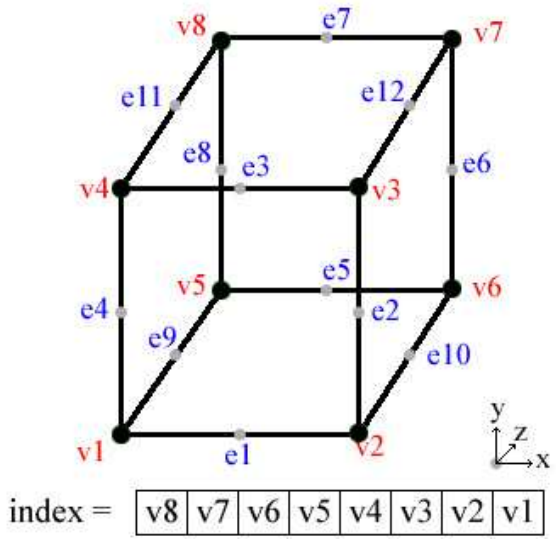


Figure 3. Index specified in the Lorensen and H. E. Cline's original article

For using such reconstructed 3D objects in a scene with lights, the normal at each vertex has to be calculated. The gradient vector is normal to the surface; it is this normalized vector, 'g' that is used.

This algorithm presents some problems, such as the high number of faces. It is now an old method, but it is still used for extracting isosurfaces. It is always used because the optimisation and mechanical aspects of the algorithm make it fast.

3.2 MARCHING TETRAHEDRON [2]

It is also possible to find the marching tetrahedron algorithm named the tetra-cubes algorithm. It is a similar technique to the marching cubes. Tetrahedral cells replace cubes. As there are only 4 vertices, there are just $2^4 = 16$ ways a tetrahedron can intersect a surface: it is less complex than marching cubes. It is simpler to implement, but the execution is slower and the isosurface extraction is less accurate (Figures 4-5). This is why it is possible to find some programs with the marching-tetrahedron and without marching-cube, or without marching-tetrahedron and with marching-cube. In fact, it just depends on the priority required; an algorithm that is fast to write and slow to execute, with a poor quality reconstruction, or an algorithm that takes a bit longer to write, but is faster to execute with a better quality of reconstruction. As this project is a student work without financial or commercial considerations, privileging quality over development time was preferred. The marching-cube, therefore, was used rather than the marching-tetrahedron.

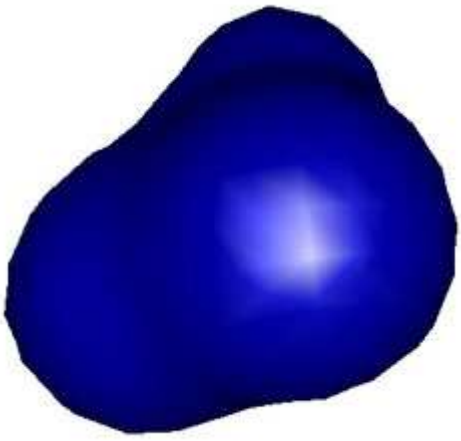


Figure 4. Marching Cube

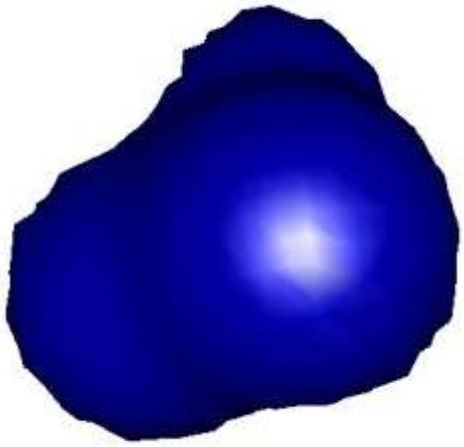


Figure 5. Marching Tetrahedron

(The same object has been reconstructed using the same resolution with two different algorithms using Cory Bloyd's Marching Cubes example program [17] found on Paul Bourke's web site[16]*)*

3.3 LAPLACIAN SMOOTHING [3]

Laplacian smoothing is the most commonly used and the simplest mesh smoothing method. The original method adjusts the location of each mesh vertex to the geometric centre of its neighbouring vertices

The method, proposed by Vollmer, Mencl and Müller, is an enhanced technique of Laplacian smoothing that avoids the problems of deformations and shrinkage. The basic idea is to push the vertices of the smoothed mesh back towards their previous locations.

3.4 DECIMATION OF TRIANGLE MESHES [4]

Schmede's decimation method principle is quite simple. Figures 6-9 show their result, which was published in 1992 in the *Siggraph*. Multiple passes are made over all vertices in the mesh. During a pass, each vertex is a candidate for removal and, if it meets the specified decimation criteria, the vertex and all triangles that use the vertex are deleted. The resulting hole in the mesh is patched by forming a local triangulation. The vertex removal process repeats, with possible adjustment of the decimation criteria, until some termination condition is met. Usually the termination criterion is specified as a percentage reduction of the original mesh (or equivalent), or as some maximum decimation value. The three steps of the algorithm are:

1. characterize the local vertex geometry and topology,
2 evaluate the decimation criteria
3. triangulate the resulting hole.

Figure 6. Full resolution (569k Gouraud shaded triangles).

Figure 7. 75% decimated (142k Gouraud shaded triangles).

Figure 8. 75% decimated (142k flat shaded triangles).

Figure 9. 90% decimated (57k flat shaded triangles).

(These pictures come from the article published in the Siggraph 1992 [4])

3.5 OCTREE-BASED DECIMATION [5]

As reconstruction algorithms produce models containing a large number of triangles, it can be interesting to use a reconstruction algorithm followed by a decimation method. Raj Shekharly, Elias Fayyad, Roni Yage and J. Fredrick Cornhill, in 1996, proposed a decimation method. Its main purpose is to reduce the number of triangles in meshes. Reduced mesh needs to preserve the topology of the original mesh and to be a good approximation of the original mesh.

To represent an isosurface, the marching-cubes algorithm generates a large number of triangles. Many triangles increase the rendering time. An Octree-based decimation algorithm can reduce the number of triangles generated by the marching cube algorithm. This algorithm

is an enhanced implementation of the marching-cubes algorithm. The decimation is performed before creating a large number of triangles. This new implementation is composed of four main steps: surface tracking, merging, crack patching and triangulation. The principle of the surface tracker is to start from a seed point, then to visit only the cells that are likely to compose part of the desired isosurface. This results in approximately 80% of computational saving. The cells making up the extracted surface are stored in an octree that is further processed. Triangles of an approximated flat surface are merged.

CHAPTER 4 - X-RAY IMAGES

4.1 HISTORY

The German Wilhelm Conrad Röntgen [18] discovered X-Rays in 1895. They are an electromagnetic radiation, emitted when matter is bombarded with fast electrons. They have a shorter wavelength than ultra-violet radiation. UV wavelenghs are about 10^{-8} metres. They extend to indefinitely short wavelengths, but below about 10^{-11} metres they are called gamma radiation. X-Rays can cross the human body. They are stopped by certain structures, such as bones, which stop X-Rays more than soft tissues. After revealing bones (Figure 10), Röntgen got the Nobel price in 1901 for the discovery of X-Rays. Since then, X-Ray images have been used in medicine as a diagnostic aid for parts of the body, such as bone.

Figure 10 Picture known as the first X-Rays image of bone (1900)

(This famous image was found on a web site dedicated to X-Ray discovery, at the following URL [18], http://www.xray.hmc.psu.edu/rci/ss1/ss1_2.html)

4.2 OTHER TECHNIQUES

Three different elements interact when a radiographer takes a traditional X-Ray image. They are: the X-Ray beam's source (or X-Ray emitter), the patient and the cassette (or X-Ray detector). X-Ray beams cross the patient and hit the cassette. The GUI has to allow users to set such a 3D virtual world. They print X-Ray images on a photographic film.

For this project, volumetric dataset are needed to get 3D models. Two main techniques allow 3D information of a part of a body to be obtained.

The first discovered, was created in 1970s[24]. It is the CT scanner, Computed Tomography scanner. An X-ray tube, rotating around a specific area of the body, delivers an appropriate amount of X radiation for the tissue being studied and takes pictures of that part of the internal anatomy from different angles. The raw signal does not form an image, a computer program is then used to form a composite, readable image.

The second technique appears in 1980[25]. It is the MR scanners. It uses the Nuclear Magnetic Resonance property. It is more recent than CT scanner, but more expensive. It produces cross-sectional images of organs and other internal body structures. The patient lies inside a large, hollow cylinder containing a strong electromagnet, which causes the nuclei of certain atoms in the body (especially those of hydrogen) to align magnetically. The patient is then subjected to radio waves, which cause the aligned nuclei to "flip" ; when the radio waves are withdrawn, the nuclei return to their original positions, emitting radio waves that are then detected by a receiver and translated into a two-dimensional picture by computer.

These two methods get 2D slices of a body. It can be considered as a 3D texture. The datasets resulting from both kinds of scanners can be used for 3D reconstructions. This project only used data obtained by CT scanners because it uses X-Ray radiations. The segmentation used in the program needs further improvement to be used with MR data. Both scanners obtain data using a spiral direction (Figure 11).

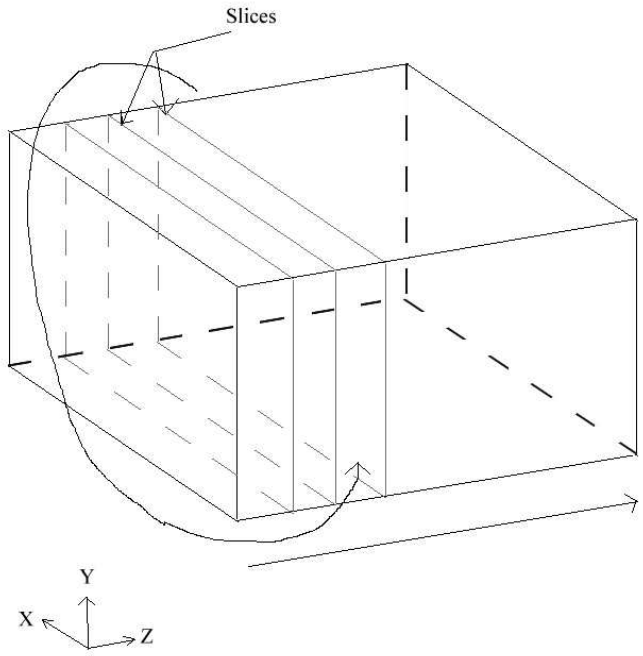


Figure 11. Scanners' moves

The MR and CT scanners move in space following a spiral. The move is in 3D. The resulting slices are not perfectly parallel because of the spiral move. They are a bit deformed, this fact is ignored during the 3D reconstruction because the deformation is not visible on the final 3D objects.

CHAPTER 5 - READING DATASET

The data needed for reconstructing 3D objects is a volumetric dataset. CT and MR scanners acquire a succession of 2D images and are able to store them into files.

5.1 THE DICOM STANDARD

The DICOM files standard is the current standard for such medical image files.

5.1.1 *Before the DICOM standard*

In the late 1970s, the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) needed a standard method for transferring images and associated information between devices manufactured by various makers. Moreover, these devices produce a variety of digital image formats. In 1983 these two organisations created a common committee (ARC-NEMA committee) to develop a standard to allow communication of digital image, regardless of device manufacturer; to develop the picture archiving and communication systems (PACS); and to create databases for diagnostics. In 1985, the first version of the ARC-NEMA standard was published, followed in 1988 by a second version.

5.1.2 *ACR-NEMA standard to DICOM standard*

The DICOM standard wants to be a general communication standard. It was done to conciliate the needs of manufacturers and users of medical imaging equipment for interconnection of devices by standard networks. The most important change from the ARC-NEMA standard to the DICOM standard is that data are no longer transferred without links between them. The DICOM standard uses a programming concept, the object oriented concept. Relationships between data are explicitly stated; this model is named E-R[8]. This new data structure is better than the previous because it shows the data items of a given scenario and how these items interact.

[8] entity-relationship

5.2 DICOM FILE

The last version of DICOM, the V3.0, gives enhancements to previous versions. Some of them are major improvements.

It is useable in a network. It supports a network environment using OSI and TCP/IP protocols. Previously, only point-to-point environments were managed.

It is structured as a document with several parts. Like a document, the standard can be enhanced in an environment that can evolve rapidly. ISO[9] directives, which define how to structure multi-part documents, have been followed in the DICOM Standard.

The most important improvement of the standard for this project, is the notion of a real file format. It appeared with the third version of the standard. A DICOM file (Appendix A) is a file with a content formatted according to the requirements of Part 10 of the DICOM Standard. In particular, such files should contain the File Meta Information and a properly formatted Data Set. Such files can contain one or several images.

5.3 DICOMIR FILES

Version 3 of DICOM deals with another kind of file; the DICOMDIR file format. It is a unique and mandatory DICOM file within a file-set which contains the Media Storage Directory SOP Class. This file is given a single component file ID, DICOMDIR (Files are identified by a File ID which is unique within the context of the file-set they belong to). In other words, the DICOMDIR files contain hierarchical information. 'DIR' of the DICOMDIR word means 'DIRectory'. Dataset information relating to patients, studies and series are stored in such files. A dataset can concern several patients. Each patient can have more than one study. A study can contain several series. A series is constituted by DICOM files ID. The figure 12 presents the tree description of DICOMDIR files.

[9] International Standards Organization

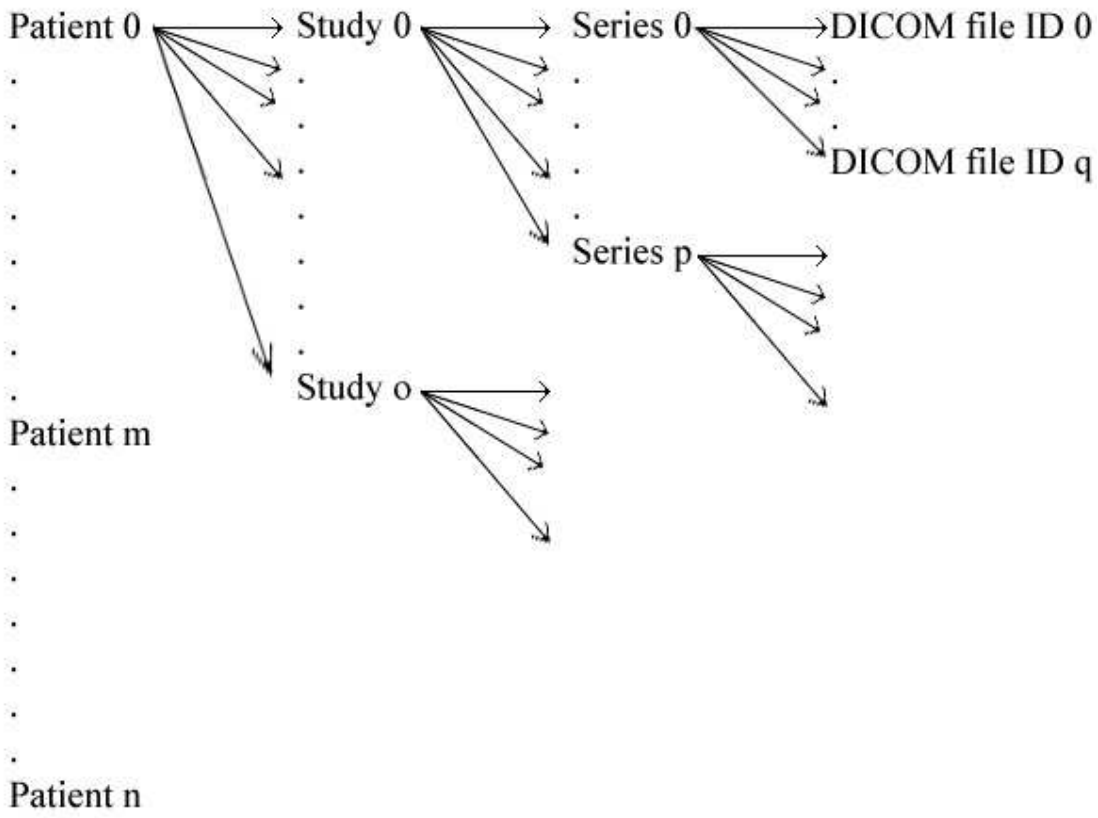


Figure 12 DICOMDIR file architecture

The dataset used during the whole development of the project is constituted by a DICOMDIR file and DICOM files containing only one image. The dataset stores information about only one patient, UOT FOOT. The patient has one study of six series.

5.4 IMPLEMENTATION ISSUES

It is not images of this dataset that have been used for the first attempt of getting images from DICOM files, because the dataset was not available at the beginning of the project. Web sites provide DICOM files for downloading. It has allowed me to test different kind of images; there were colour or grey scale images, lossy or not compressed data, 16 bits or 8 bits encoding.

A C++ library, named DiLib ("DI" for DICOM and "LIB" for library) is implemented to read and manage DICOM and DICOMDIR. It is written in C++ on the top of the Papyrus toolkit, because both file formats need the same Papyrus functions. Figure 13 shows the architecture of the library implemented to manage datasets using DICOM and DICOMDIR files.

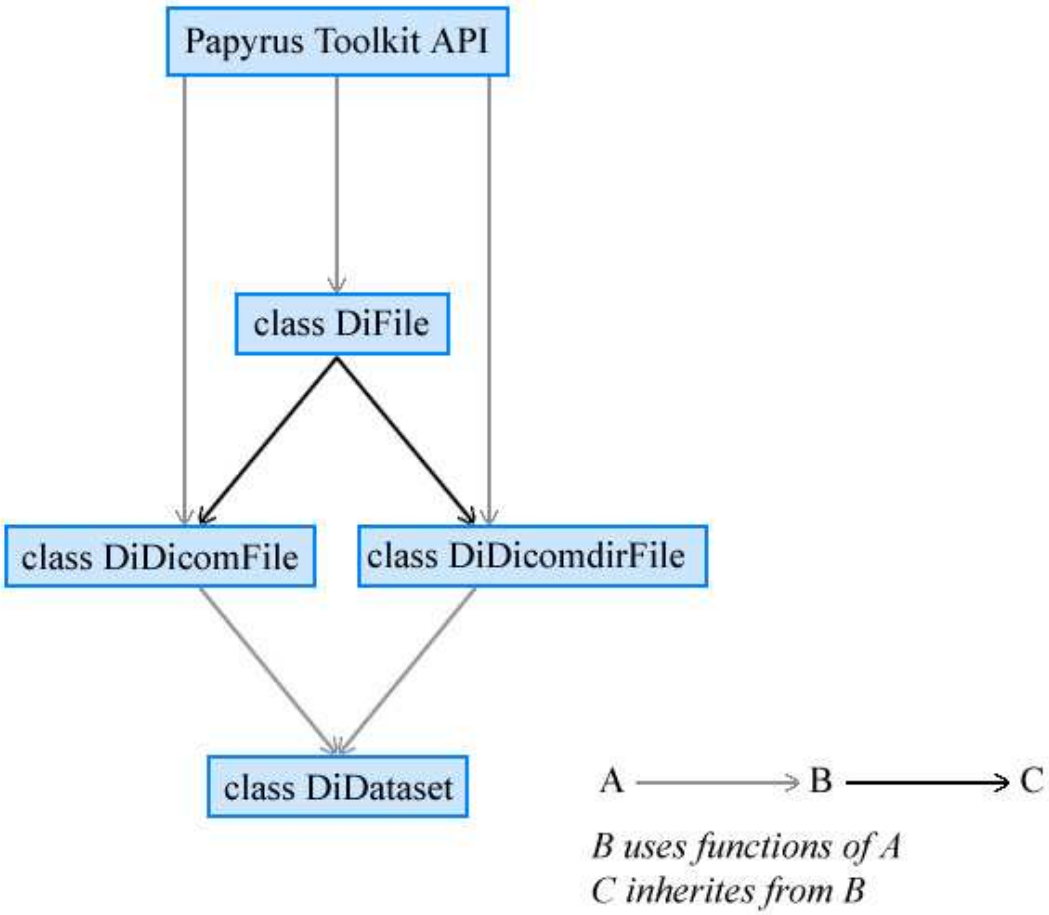


Figure 13 DiLib architecture

CHAPTER 6 - 3D RECONSTRUCTION – IMPLEMENTATION ISSUES

6.1 IMAGE PROCESSING

6.1.1 *Segmentation*

Before creating any 3D surfaces and volume meshes from CT images, an algorithm has to determine the nature of each pixel. The purpose of the image segmentation for medical imaging applications is the construction of a series of regions usually called ROI[10]. For this project, a pixel can represented skin, bones, air or tissue. Another range of pixel values has to be managed. Scanners produce circle images. An image is stored in file as a rectangle. To obtain a rectangle from a circle, each corner is filled by a unique value, named 'Padding value'. Pixels corresponding to the padding value are replaced by black pixel value, 0. As the program uses the Marching-cube algorithm, the segmentation algorithm has only to determine which pixels are outside, or inside the body, according to the required object, tissue or bone. The image segmentation creates a new image with a unique intensity for each kind of pixel value.

The program, in fact, performs two successive reconstructions when a dataset is opened. One corresponds to the soft tissue model and the other one to the bone model. Figures 14 and 15 illustrate the segmentation for the soft tissue model. For such segmentation, bones are ignored; the program considers bones as tissue because they are inside the body.

[10] region of interests

Figure 14. CT image

Figure 15. Image after segmentation

The used segmentation algorithm only tests pixel values. From the CT image, a very small pixel value corresponds to air, a very high to bone and a medium value to tissue. These pixel values are the consequence of the fact that the denser a material is, the more it absorbs X-Rays.

Two different threshold values are defined to perform to different segmentations, one for the skin and the other one for the bone. The lower value is the border between air and skin, the second, is the border between skin and bone. Each pixel has to be tested; if a pixel is inferior to the lower step, it is air; if it is between the two threshold values, it is tissue; if it is superior to the upper threshold value, it is bone. For the soft tissue model, all pixel values superior to the lower threshold value are considered as soft tissue. This piece of code illustrates the algorithm used. It is a threshold algorithm.

```
UCHAR* tmp_target_buffer = _BufferSegmentedImage;
UCHAR* tmp_source_buffer = _Buffer8BitsColorDepth;

// Test each pixel
for (int i = image_width * image_height; i--;)
{
      if (*tmp_source_buffer++ <= THRESHOLD_VALUE)
            *tmp_target_buffer++ = WHITE;
      else
            *tmp_target_buffer++ = BLACK;
}
```

Another method of performing the segmentation is to use a histogram (Figure 16).
It will allow the detection of different ranges of pixels. The histogram counts the number of pixels for each pixel value of an image.

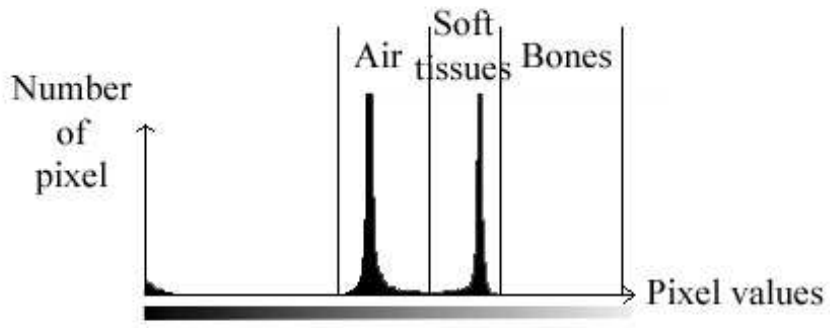


Figure 16. Histogram, obtained by Photoshop[27], of the image of Figure 14

On the histogram of the image of Figure 14, there are four different ranges of pixel values. On Figure 14, there are:

- the range on the left corresponding to the black corners of the source image
- at the right of this range, there is a big range which corresponds to air
- at the right of this range, there is a big range which corresponds to soft tissues
- at the right of this range, there is an important range that corresponds to bones. The few number of pixels corresponding to bones does not show the last range pixel value, but it exists.

6.1.2 *Removing little artefacts*

2D slices can have some artefacts. Their size can be small or big. On figure 17, the red lines can be automatically removed because the width of the line is about 5 pixels. The green marked artefacts cannot be deleted because they are too big. If such artefacts are removed automatically, some detail of the foot, such as the toes, will also be removed; it is not what the program is supposed to do. Only little artefacts can be removed.

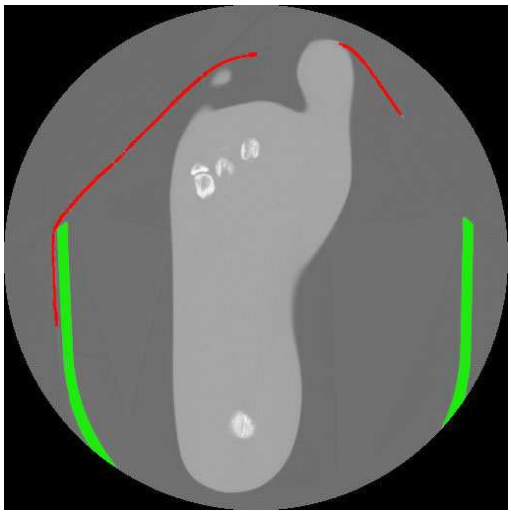


Figure 17 Little artefacts

The classic way to automatically remove this kind of small artefact is to use an 'open' algorithm, followed by a 'close' algorithm. An «erosion» algorithm followed by a «dilatation» algorithm composes the «open» algorithm. A «dilatation» algorithm followed by an «erosion» algorithm composes the «close» algorithm. Appendix B shows the effect of these four algorithms.

Such methods cannot be used because of the time of calculation. A method has been found to remove small artefacts. It is not as clean as the open/close algorithm, but it is faster. This algorithm can only be applied on a segmented image. Its purpose is to seek small surface defects (surfaces of white pixels) and to remove them (by replacing white pixels by black pixels). The research of such artefacts is only done in horizontal and vertical lines; it is why this algorithm is faster than the open/close algorithm, which does care about all neighbouring pixels. Changes of pixel values are detected on lines and are removed. The algorithm seeks changes from a black area, to a very thin white area, to black area. Figure 18 illustrates how it is performed.

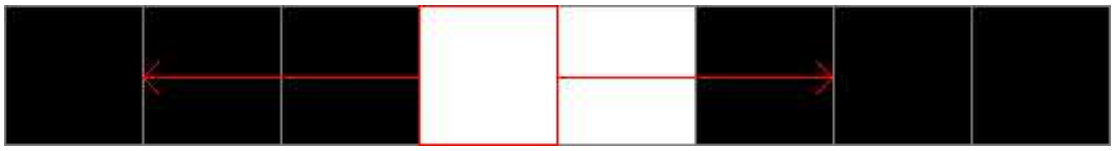


Figure 18. Principle of removing artefact algorithm

This example reduces the matter to a horizontal line. A value, defined by the user, specifies the number of pixels for the research of artefacts. In this case, it is two. For

a white current pixel, the program looks for one of the two (the defined value) pixels of the current pixel, which is black. The same is done on the right. If a black pixel has been found on the right and one pixel has been found on the left, the program replaces the white current pixel by white. If it is not the case, the program tests pixels, following a vertical line. The following figure shows the result of the algorithm on the same picture as figures 14 and 15.

Figure 19. Little artefacts have been removed

6.2 3D RECONSTRUCTION USING MARCHING CUBE

The step following the image segmentation is the obtaining of a 3D object, which can be manipulated in the 3D space by the user.

6.2.1 *Why the marching-cube*

For several reasons the choice is made to convert the surface of the 3D object in a triangle-mesh. Volumetric rendering is too slow on a current computer. The program, ***MedicView*** [12] allows two kinds of rendering, volumetric rendering and mesh rendering. ***MedicView*** has been tested on a common computer (AMD Atlon 1200MHz with a Nvidia GeForce2 MX and 256 Mo of RAM). The object is difficult to manipulate with volumetric rendering, because the execution speed is low. With the mesh it is faster.

For converting the surface to a triangle mesh, two main algorithms are available, the marching-cube and the marching-tetrahedron. As figures 4 and 5 show, the marching-tetrahedron produces more inaccuracies than the marching-cube, but its advantage on the marching-cube is that it is easier to implement. Yet another important negative point of this algorithm is that it is slower than the marching-cube. The advantage of the marching-tetrahedron is not enough, to ignore the older algorithm; the faster algorithm, which produces a more accurate reconstruction, the marching-cube, has been chosen.

6.2.2 *Reconstruction of the surface of the 3D object*

The algorithm tries to locate the surface of the object in a logical cube, created from four pixels of a slice and four pixels of the adjacent slice. If a pixel value is less important than a threshold, the pixel is outside the surface of the 3D object. If the pixel value is equal or superior to the threshold, the pixel is on or inside the surface. The threshold is commonly called isovalue. A cube has eight vertices and each vertex can have two different states, inside or outside, just eight bits are needed. A bit is a binary value, which can only be equal to 0 or 1. A byte contains 8 bits. Just a byte is required to store the state of all vertices of the cube. It was decided that if a vertex is outside, the corresponding bit is set to 0, and it is set to 1 if it is inside, as it is specified in the original Lorensen's article. One bit of the byte corresponds to the state of one vertex of the cube. Figure 20 illustrates the order of the vertices on the

cube and figure 3 shows the states of the vertices' order in the byte. The following code allows the byte containing the cube vertices' states to be filled.

```
#define DI_V0   1 // = 2^0 = 0000 0001
#define DI_V1   2 // = 2^1 = 0000 0010
#define DI_V2   4 // = 2^2 = 0000 0100
#define DI_V3   8 // = 2^3 = 0000 1000
#define DI_V4  16 // = 2^4 = 0001 0000
#define DI_V5  32 // = 2^5 = 0010 0000
#define DI_V6  64 // = 2^6 = 0100 0000
#define DI_V7 128 // = 2^7 = 1000 0000

/* Find which vertices are inside or outside of the surface
 *
 * - put a 1 to the bit if the vertex is on or inside of the
 * surface
 * - put a 0 to the bit if the vertex is outside of the surface
 */
// Fill the byte with zeros
cube_description = 0;

if (*p_first_slice >= THRESHOLD) cube_description += DI_V0;
p_first_slice += cube_width;

if (*p_first_slice >= THRESHOLD) cube_description += DI_V1;

p_temp1 = p_first_slice + x_step1;
if (*p_temp1 >= THRESHOLD) cube_description += DI_V3;
p_temp1 += cube_width;

if (*p_temp1 >= THRESHOLD) cube_description += DI_V2;

if (*p_second_slice >= THRESHOLD) cube_description += DI_V4;
p_second_slice += cube_width;

if (*p_second_slice >= THRESHOLD) cube_description += DI_V5;

p_temp2 = p_second_slice + x_step1;
if (*p_temp2 >= THRESHOLD) cube_description += DI_V7;
p_temp2 += cube_width;

if (*p_temp2 >= THRESHOLD) cube_description += DI_V6;
```

By using masks, any bit of the byte can be filled.

The following instruction puts zero in each bit of the bytes.

```
cube_description = 0;
```

It becomes:

0	0	0	0	0	0	0	0

By adding a value to the cube_description variable, the value of the wanted bit was changed, for example, if the vertex 0 intersects the surface, I will add DI_V0.

```
cube_description += DI_V0;
```

It becomes:

0	0	0	0	0	0	0	1

If the vertex 3 also intersects the surface, I will add DI_V3.

```
cube_description += DI_V3;
```

It becomes:

0	0	0	0	1	0	0	1

If the vertex 5 also intersects the surface, I will add DI_V5.

```
cube_description += DI_V5;
```

It becomes:

0	0	1	0	1	0	0	1

Now the cube_description is equal to $2^0 + 2^3 + 2^5 = 41$

As a cube has 8 vertices that can have two different values, there are $8^2 = 256$ different cubes. It is possible to write my own lookup table, but it is very easy to make mistakes. Moreover, it is already done and available on the Internet. The table that I have used has been found on the Bourke's Internet site (from the Swinburne University of Technology in Australia) [16]. According to the lookup table, the cube edges and vertices order is the following:

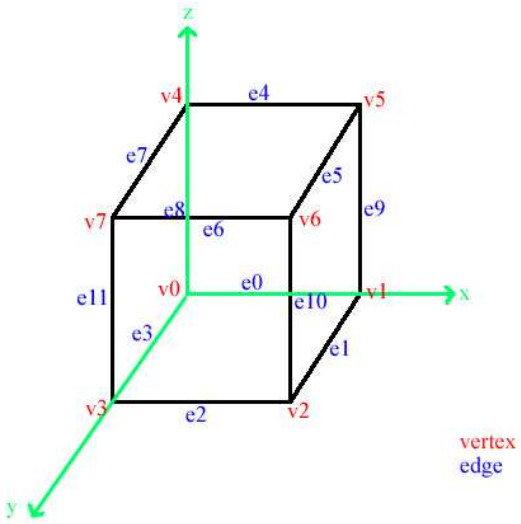


Figure 20 Cube edges and vertices index

After seeking which vertices are on the surface or not, I obtain a byte. Its value varies from 0 to 255. It gives me an index to find in the lookup table if an edge of the cube

intersects the surface of the 3D object. At least one edge intersects the surface if at least one vertex of the cube is on the surface; but if all vertices are on the surface, no edge intersects the surface because the cube is inside the 3D object. If the surface intersects the cube, I seek triangles that have to be reconstructed in a lookup table using the same byte as index. This lookup table is composed by 256 * 16 elements (256 because there are 256 different cubes and 16 because a cube may have five triangles and a triangle is composed by three vertices). The following piece of code is the five first lines of this lookup table.

```
GLbyte g_triangle_connection_table[256 * 16] =
{
      -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       0,  8,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       0,  1,  9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       1,  8,  3,  9,  8,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       1,  2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
.
.
.
```

To consider the third line of the example, that is to say 0, 1, 9, -1, … To access this line, the cube vertices description byte is equal to 2, that is to say that only the vertex 1 is on the surface. For reconstructing the surface at this cube, a triangle has to be drawn using the edges 0, 1 and 9 like the figure 21 shows. It is the values found at the considered line.

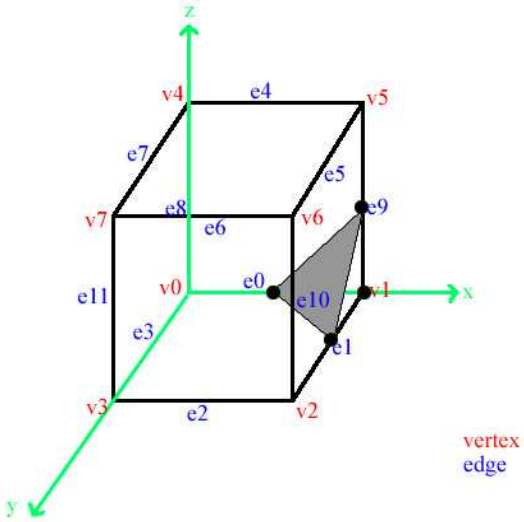


Figure 21 The vertex 1 intersects the surface

To reconstruct the whole surface of the object, the cube is marched through the whole volumetric data using three loops. To do it, the following loops can be used:

```
// Use all slice
for (z = 1; z < _NumberOfSlice; z++)
{
      // Load the next slice
      ...

      // Use all rows of the image
      for (y = 1; y < _ImageHeight; y++)
      {
            // Use all pixel of the current row of the image
            for (x = 1; x < _ImageWidth; x++)
            {
                  /* Test cube vertices and draw triangle
                   * if it is needed
                   */
                  ...

                  // March the cube
                  ...
            }
      }
}
```

6.2.3 *Optimisations*

CT and MR scanners get volumetric data by turning around the scanned object. The way followed by scanners is a spiral. Produced slices are not parallel. The result of the reconstruction shows that it is not needed to have a parallel slice if the space between two successive slices is not too big.

As the 3D objects do not change during the setting of the radiography, the program can store them into the memory and does not have to reconstruct them each time that the screen is refreshed. Moreover, as the 3D reconstruction takes a long time, the 3D objects have to be stored in memory. 3D objects are a set of triangles. Three vertices compose each triangle. Each vertex is composed by its coordinates, that is to say, three floats. To reduce the size in memory as much as possible, an array of vertices is needed for each object where vertices are unique (the same vertex is stored once). To reduce the size of the first array, each vertex is stored once. To get triangles from this array, another array is needed. It stores a succession of triangles by referencing three vertices by triangle (Figure 22).

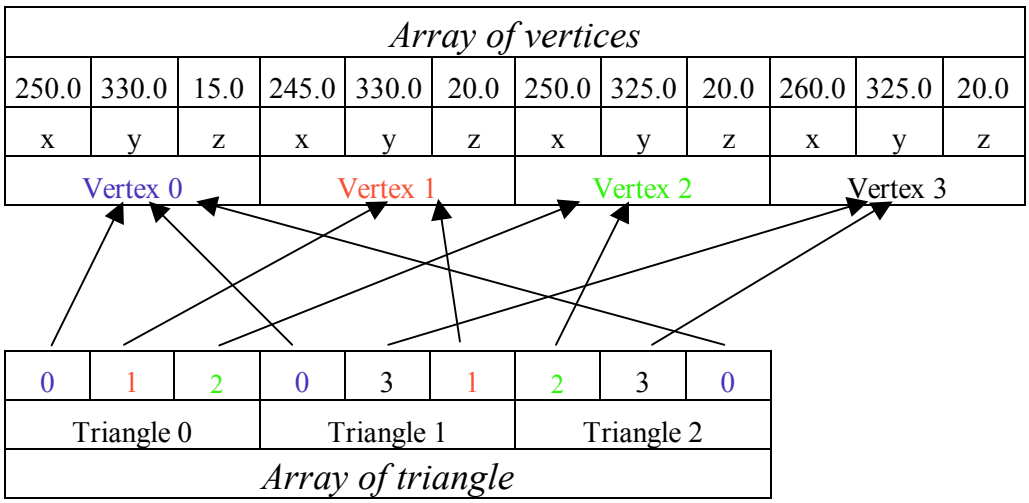


Figure 22 Example of the vertex and index arrays of a 3D object

It is impossible to predict the number of vertices and triangles before carrying out the marching-cube. As the size of an array is fixed, one can be used. The only data structure that can be used to store the vertices of the 3D object during the 3D reconstruction, is a linked-list (Figure 23) because the size of the list is not fixed. As node as it is required can be added to the linked-list, the only limitation is the memory size. When a new vertex needs adding to the list, a new node is created. The implementation of the marching-cube uses two lists, one for the vertices and the other one for the triangles. When the program has got the 3D object, it creates two arrays and copies the linked-lists into the corresponding arrays.

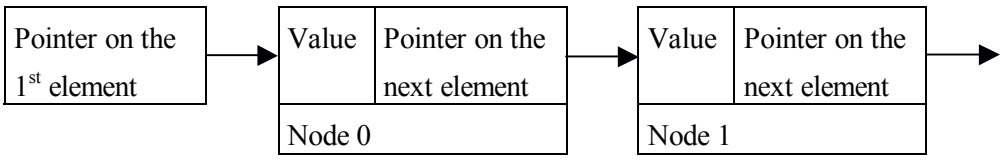


Figure 23 Linked-list

For reducing the size of the vertices' linked-list during the reconstruction and the size of the corresponding final array, the same vertex cannot be found twice in the array. For this, a check is needed to see if a vertex is already on or not on the linked-list. Normally each vertex of the list has to be tested and the three components of the vertices compared. A method is used for optimising this point. The linked-list is sorted, firstly on x, then y and then z. Like that, not all vertices of the whole list are tested; a sorted insertion is carried out.

The same optimisation could be done twice using a double linked-list. For a double linked-list, each node contains a pointer on the previous node. For the insertion of a node, the average of the x-component of the first and the last node is calculated. If the x-component of the node to be inserted, is nearer the value of the x-component of the first node, vertices are tested from the beginning, else from the end. The reconstruction should be much faster than with the use of the previous method.

The marching cube uses image processing. Image processing algorithms are generally very expensive to execute. To go through a whole greyscale image the basic method is as follows:

```
// Declaration of the image as a 2D area
unsigned char p_image[image_height][image_width];

unsigned char pixel_value;
for (int i = 0; i < image_height; i++)
      for (int j = 0; j < image_width; j++)
            pixel_value = p_image[i][j];
```

As the image is defined as a 2D area, the number of processor instructions to get a value is really important. It needs multiplications, which are slower than additions. It can be reduced by using a 1D area and a 'reverse loop'.

```
unsigned char p_image[image_width * image_height];
unsigned char* p_tmp = image;
unsigned char pixel_value;
for (int i = image_width * image_height; i--;)
      pixel_value = *p_tmp++;
```

Some values are needed several times during the execution of the marching cube algorithm. Some of them do not change. They can be stored in memory, as the program does not have to calculate them each time that is needed.

```
GLubyte nb_pixel_not_used_per_row((_CurrentWidth - 1) %
      cube_width);
GLubyte nb_pixel_not_used_per_colon((_CurrentHeight - 1) %
```

```
      cube_height);
GLubyte nb_pixel_not_used_per_slice((_CurrentDepth - 1) %
      cube_depth);
```

These values are needed each time that the cube moves, but they never change. If the program calculates them each time, it will become long.

To get the byte, which describes the vertices' cube, additions are used, not bit shifting, because, the mesh is reconstructed once, not in real time. Using additions, this part of the algorithm is slower than using bit shifting, but easier to read.

Bit shifting is only used for dividing an integer (int, which is 32 bits variable, or short (16 bits) or char (8 bits)) by two because it is easy to read ((i >> 1) and (i / 2) are the same thing) and it is much faster. A multiplication of numbers by 2 are replaced by a sum. (i + i) is faster than (i * 2) because addition is faster than multiplication.

6.2.4 *Reducing the LOD*[11]

The previous example shows the three loops for performing the marching-cube with the smallest cube possible, that is to say, with the biggest resolution. It produces a lot of faces, too much for current computers.

Generally, for the marching-cube used with volumetric data as a succession of 2D slices, a cube marches between each pixel of the 2D pictures. For the dataset of the phantom of the foot, the reconstructed object of the soft tissue (Figure 29) is composed of about 500 000 triangles and 450 000 vertices. It is the same for the reconstructed object of the bones (Figure 30). It is really too much for a current computer. The result looks good, but the manipulation of 3D objects in the space becomes difficult, because it is too slow. Moreover, the 3D reconstruction is very slow. At least 24 hours are needed to get the skin model and the bone model (the reconstruction is so long because of the use of a linked list).

A choice has been made between two methods. The first method is to grow the number of pixels per cube (Figure 24). The second method is to reduce the size of each slice. This method is quite simple in 2D. For a slice, only some pixels are considered and interpolated, but in 3D it becomes more complicated, because the interpolation is done between different slices. If such an algorithm is done, the reconstruction becomes very long.

[11] Level Of Detail

The following figure illustrates how the LOD was reduced; a bigger cube size is used, some pixels are ignored.

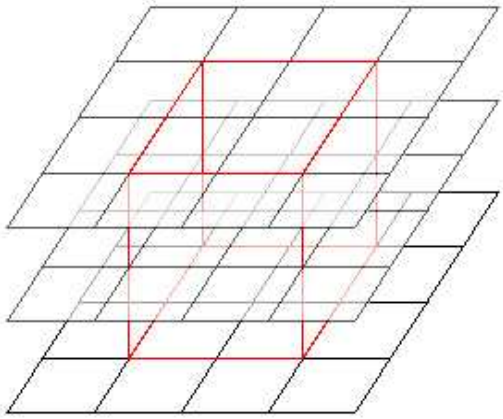


Figure 24. Bigger cube size principle

As the result on the screen looks acceptable enough (Figure 25), this method has been retained.

Figure 25 Increasing the cube size

The visible result shows something else. The little red lines of figure 17 are no longer a problem. Just a few artefacts appear. That is why the algorithm for deleting little artefacts is disabled; it was decided that it is more important to get a fast reconstruction than removing all the little artefacts. The calculation time decreases.

Another method reduces the number of artefacts of the reconstructed model. Some areas can be ignored if the user selects some regions on an image where the pixel values are not used during the reconstruction.

6.3 NORMAL

To get a 3D model, in a scene with lights, the program has to calculate normal. Two ways are possible to achieve this. There can only be one normal by face, or one normal by vertex (three by face).

Calculating normal by face is done by using the cross product between two vectors that define a triangle. At the end, triangles are too much visible.

The second method produces a more realistic result. Figure 26 has been obtained using this method; it produces a Gouraud shaded mesh. Getting a unit normal vector for each triangle vertex, consists of calculating the gradient vector, because the direction of the gradient vector is normal to the surface.

The gradient is computed like this:

$$G_x(i, j, k) = (D(i + 1, j, k) - D(i - 1, j, k)) / d_x$$
$$G_y(i, j, k) = (D(i, j + 1, k) - D(i, j - 1, k)) / d_y$$
$$G_z(i, j, k) = (D(i, j, k + 1) - D(i, j, k - 1)) / d_z$$

With D(i, j, k) the density at pixel(i, j) in slice k.
With d_x, d_y and d_z the lengths of the cube edges.

The program has to store and manipulate four successive slices at a time. The first and the last slices are used only for the normal computation.

Figure 26 Gouraud shading

As vertices obtained by marching cube are on the middle vertex of the cube edges and the method for calculating normal gives normal at cube vertices, the algorithm has to calculate eight normal by cube (Figure 27) and interpolate them to get the required normal (Figure 28).

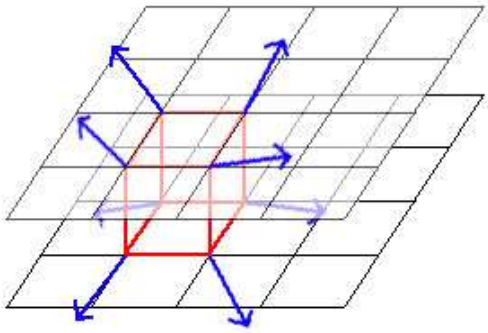


Figure 27 Normal at each cube vertices

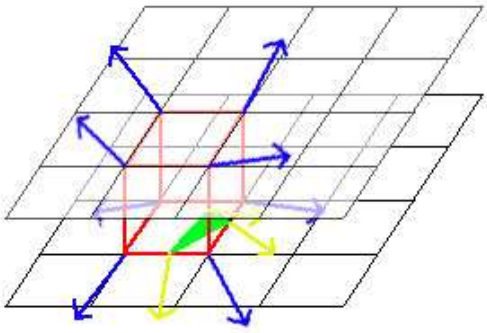
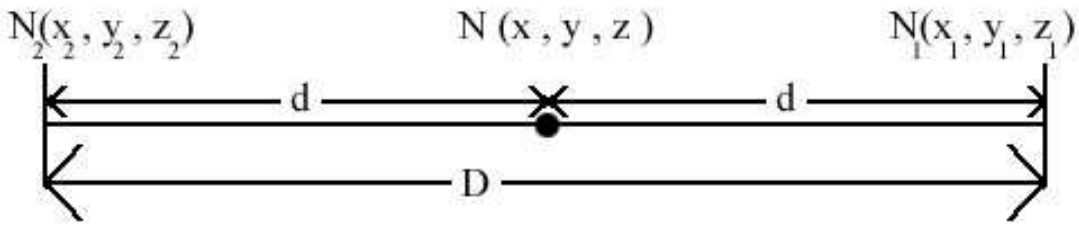


Figure 28 Linear interpolation of normal

The algorithm of the interpolation is as follows:



N1 and N2 are the normal at vertices of the same edge of a cube.

N is the normal at the middle edge:

$$x = x_1 + (x_2 - x_1) * D/d = x_1 + (x_2 - x_1) * 0.5$$

$$y = y_1 + (y_2 - y_1) * D/d = y_1 + (y_2 - y_1) * 0.5$$
$$z = z_1 + (z_2 - z_1) * D/d = z_1 + (z_2 - z_1) * 0.5$$

As the image processing algorithm used is applied in 3D, it is slow; that is why the program is optimised by the used of backup variables. Each value used several times is stored into memory. The program becomes less easy to read, in spite of the use of characteristic variable names, but it becomes much faster.

6.4 TEXTURING

The photo-realistic aspect of the reconstructed object is not a priority. The real-time manipulation of 3D objects is the priority. As the use of light means a lot of calculating time, it was decided that it was not necessary to slow down the program by using texture mapping. The use of textures would increase the photo-realistic aspect, but it was preferred use just two surface materials corresponding to the type of 3D object. Determining the ambient, the diffuse and the specular reflectance of the materials can simulate material properties of a surface. For the skin surface, the material used imitates the colour of human skin with a realistic property of shininess and specular component (Figure 29); the bones are represented by a matte white effect (Figure 30).

Figure 29 Skin material

Figure 30 Bone material

CHAPTER 7 - GRAPHICAL USER INTERFACE

For setting the scene with these reconstructed objects, the X-Ray source and the cassette, a user-friendly GUI is needed. Philip Cosson, who proposed this project, would like to have a program that works with different Microsoft operating systems. Therefore, the API chosen is the Microsoft's Win32 API. Appendix C shows the main window of the program.

7.1 GUI ARCHITECTURE

The use of the program is divided into three different steps. The first is the obtaining of the 3D objects. The second is the setting of the scene. The last is the execution of the X-Ray rendering program.

The following schema presents the architecture of the GUI with all different main events.

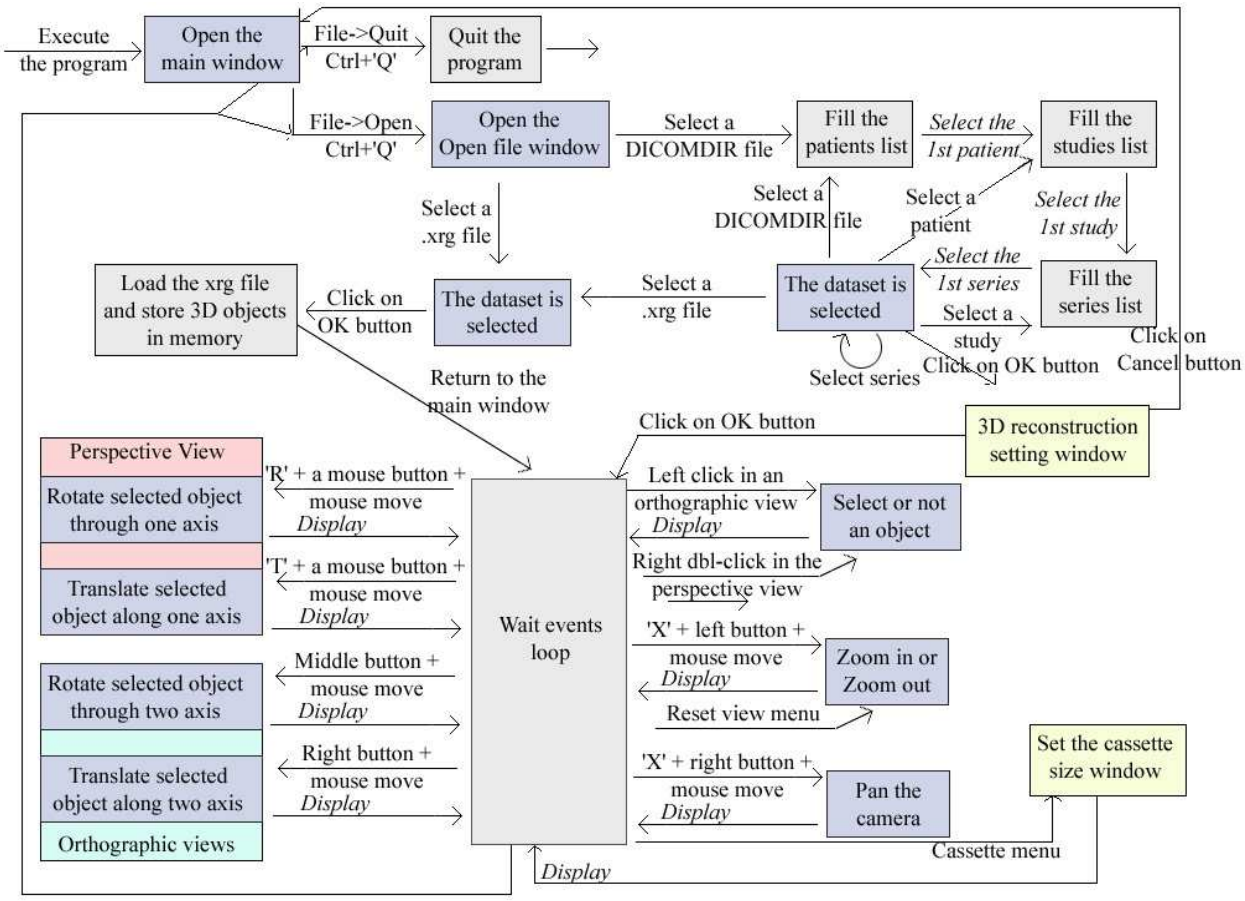


Figure 31. GUI events

7.2 SETTING THE 3D RECONSTRUCTION

The 3D reconstruction needs parameters. The cubes' sizes, the two thresholds for the segmentation, require user input.

7.2.1 *Setting the segmentation parameters*

A window is dedicated to the 3D reconstruction setting. The user can set the size of the two cubes, one for the skin model and the other for the bone model.

Two sliders allow the user to set the threshold. The user can see in real-time the corresponding segmentation, but as the value can be between 0 and 65535, the sliders are not accurate enough. Textboxes should be added if there was time, because the users should be able to set any value that they require. The original image is displayed near the segmented image. Users can change the loaded slice with a 'spin control'.

Appendix C shows the window used.

7.2.2 *Reducing the 3D object's loading time*

The 3D reconstruction can take a long time. It depends on the required resolution. The same reconstructed object can be used several times, therefore, it can be stored in a file. To test the marching-cube, it was decided to store reconstructed objects in an ASCII file format of 3DS Max[26]. 3D file viewers allow this kind of file to be visualised. The shareware used is 3D Exploration[22] because it reads many different file formats. Such files can be opened in a text editor, which is useful for debugging.

It has been decided to change the file format for several reasons. ASCII files are bigger than binary files. As it is bigger, more time is required to load them. That is why a binary file format is used. An existing file format could be used, but for space optimisation, it has been decided to store only what the program requires. It required the Patient ID, the Study ID, the Series ID (corresponding to a DICOMDIR file), the skin mesh and the bone mesh. As the program needs specific information, such as the different IDs, a file format has to be created for this project. Appendix D shows the file format created for the GUI, the file extension of this file is 'XRG' for X-Ray GUI.

Such files have to be located in the same directory as the DICOMDIR file, otherwise, the program should ask where the DICOMDIR file is, to know where the dataset is for sending this parameter to the other program. But as I am not able to read DICOMDIR files, the program asks for the first and the last DICOM files of the dataset.

When the 3D objects are loaded from XRG files or are reconstructed from the dataset, the 3D objects are integrated in a virtual world.

7.3 GETTING PROPORTIONAL SIZES

The different 3D objects, which composed the scene, have to have proportional sizes. Information, in millimetres, is saved in DICOM files. The unit chosen is the millimetre because of this fact.

The cassette, or X-Ray receptor, is just a cube. Its value can be chosen from a menu because there are standard sizes. If users needs a custom size, they are able to enter their own parameters in a window.

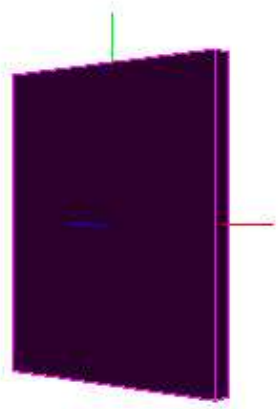


Figure 32. The cassette

The X-Ray source, or X-Ray emitter, uses two wire frame cubes for the ceiling tracks and a solid cube for the vertical column. A solid sphere is located at the bottom of the column. A line shows the direction of the X-Ray's beam.

The longer of the ceiling tracks is 3 meters. The space between lines is 10 centimetres. The radius of the sphere is 20 centimetres.

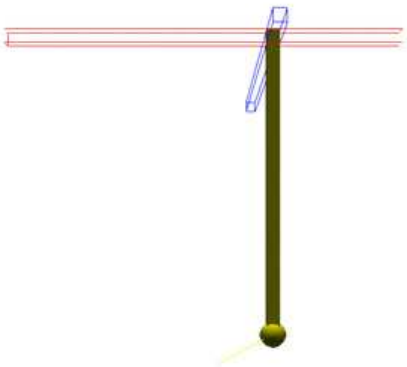


Figure 33. The X-Ray source

The DICOM files store information about the space between two adjacent pixels, the space between two successive slices and the thickness of slice. DICOM part 3 describes these parameters. Using them during the reconstruction, the cube size in pixels is modulated to get a size in millimetres. Depending on the dataset used, the result do not always look like what was expected. More time is required to make the method more accurate.

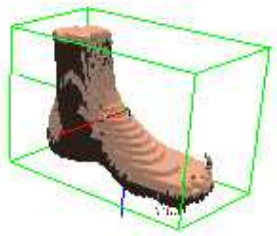


Figure 34. The patient

7.4 SETTING POSITIONS AND ORIENTATIONS

The easier way to set a 3D space is to have at least three orthographic views. That is why the GUI was given three orthographic views; top, bottom and left. A fourth view, which simulates a more realistic view, is perspective. Figure 35 shows how the world is set up.

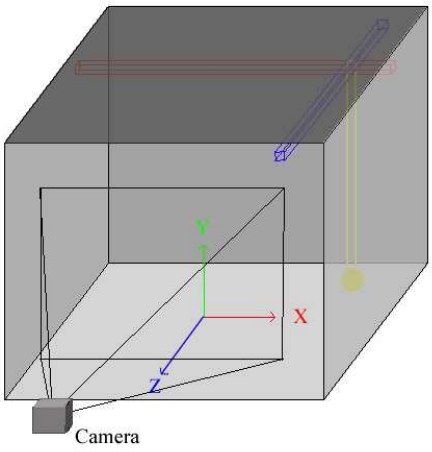


Figure 35. The world with the perspective camera

As there are three different objects in the scene, the user has to select an object before rotations and translations. Two different methods can be used to select an object. In orthographic views, the user can select an object just by clicking on it with mouse using the left button (as in any Windows program). To do this, the mouse pointer coordinates are translated from screen in pixels to millimetres. This conversion depends on the zoom factor of the view where the user has clicked and the kind of the view (top, bottom, front, back, left or right). In the perspective view, the user can double-click anywhere with the right mouse to change the selected object.

To be easier and more user-friendly, it is possible to zoom-in and out and pan the camera to any view. To do this, the user must press and hold 'X' on the keyboard, and use the left button to zoom, or the right to pan, in the same manner as SoftImage[23].

7.4.1 *Cassette and patients*

In the orthographic view, the user can translate the selected object under the cursor position, if he presses and holds the right mouse button and moves the mouse. It cannot be more user-friendly.

The rotation is inspired by the internet browser plugins of Viewpoint Corporation[24]. These plugins allows the user to manipulate 3D objects with the internet browser, using the mouse. The plugins do the rotation through two local angles. But the plugins manage only one view. The same technique is done in the different orthographic views. The local axis chosen depends on the view.

For top and bottom views, a horizontal move of the mouse applies a rotation through the z-axis, a vertical through the x-axis.

For front and back views, a horizontal move of the mouse applies a rotation through the y-axis, a vertical through the x-axis.

For left and right views, a horizontal move of the mouse applies a rotation through the y-axis, a vertical through the z-axis.

7.4.2 *X-ray beam sources*

The X-Ray source can be selected in an orthographic view only if the user clicks on the sphere. It is impossible to move an actual X-Ray machine by moving the column or a ceiling track; it is the same thing with the GUI.

In the orthographic view, the user can move the X-Ray emitter using the same method as for moving the cassette or the X-Ray source. The same method as the other objects' rotation is used for the X-Ray emitter. The result is not perfect. Although this method produces a user-friendly rotation for the patient and the cassette, it is not the case with the X-Ray source. This difference comes from the fact that the X-Ray source is just represented by a line. More time is needed to create a more user-friendly rotation of the X-Ray source.

The control of objects in the perspective view is different. It is possible to move objects in the perspective view, using the mouse and the keyboard. It is easy to move any object. The translation along the x and the y axis is performed by pressing and holding the 'T' key and using the left mouse button. For a translation along the z-axis, the right button is used.

For rotating objects, it is possible to press and hold the 'R' key, and use the left button to rotate around the x-axis, the middle to rotate around the y-axis and the right button to rotate around the z-axis.

7.5 THE LINK BETWEEN THE GUI AND THE X-RAY RENDERER

The GUI sets some parameters needed by the volumetric rendering program; the name of the file of the dataset, the patient, the study and the series indices, the position and the rotation of the patient, the cassette and the X-Ray tube and the size of the cassette have to be known by the renderer. Three different ways are possible for the companion program and the GUI program to work together.

Theoretically, communication between processes is possible, but, as the two programs have been developed independently by two different programmers, it is difficult to implement the communication and test it in the given timetable. Moreover, this method makes programs dependent on each other, because shared memory between processes has to be used. The shared memory has to be created by only one process. Such a consequence makes the testing of each program difficult.

Another method is the use of a temporary configuration file written by the GUI program and read by the rendering program. After creating the file, the GUI can create a new process from the executable file of the companion program, or the user can create it via a OS[12] command (as the console, or a shortcut under MS-Windows).

The last method consists of passing information to the renderer via the command line arguments. The rendering program has to read and translate a string obtained by its 'main' function of the program.

Example of use of the command line arguments: *file.exe arg0 arg1 arg2 arg3*

The agreed protocol is the last one, because it is easy for the two programmers to use and is fast to implement. The programs are totally independent; it is easy to test a program without the other. The GUI creates a child process from the executable of the renderer and passes required arguments by the command line arguments. Appendix E illustrates the order of parameters and how to use command line arguments when a program creates a child process.

[12] Operating System

CHAPTER 8 - PROGRAM IMPLEMENTATION DESIGN

8.1 GENERAL DESIGN

As sharing code between different programs is an important aspect of programming, the program design of any program must be thought. It was decided to divided the program into four parts (Figure 36). The DICOM and DICOMDIR file reading functions are part of a library. A core library has been used, consisting of a C++ class which manages strings, template C++ classes for managing linked-lists and mathematical functions. Another part is dedicated to the graphical aspects, the windows management and 3D. The last one is the most specific code of the program, it manages all different windows and events of the interface. The purpose of libraries is to deal with functions. These functions are common to different programs, libraries illustrate the important aspect of sharing code between programs. In this code, there are three libraries (Figure 36).

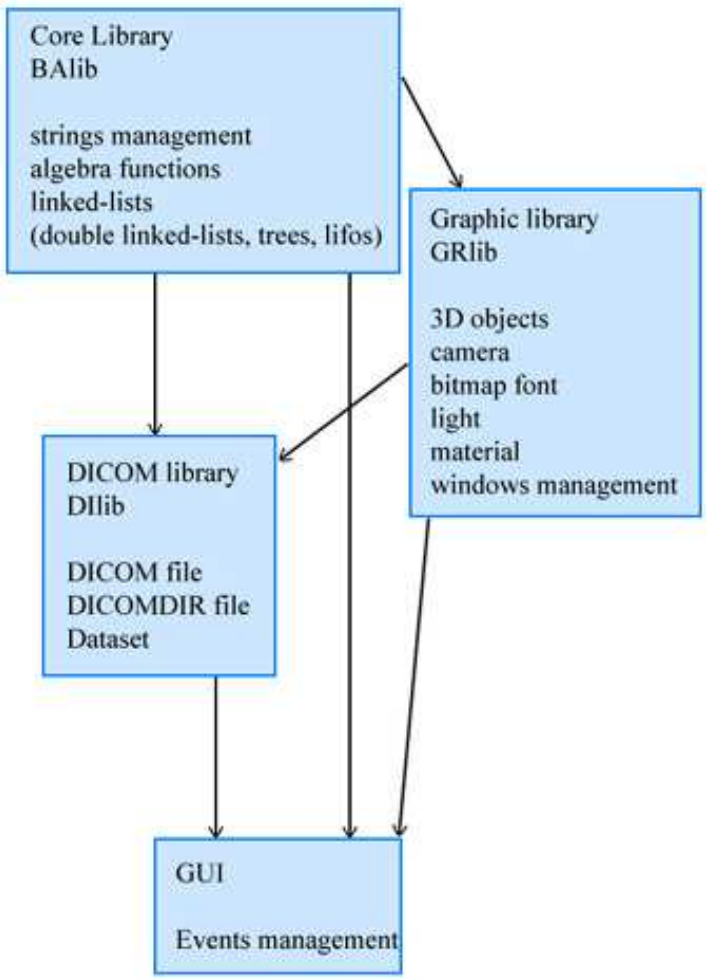


Figure 36. General implementation design

8.2 CLASSES AND LIBRARIES

As my program is the first step of the development of another program, which will use VR devices, the libraries can be included in future programs for dealing with DICOM files and marching, for instance.

The programming language chosen for writing this program is the C++. It is an oriented object language. It is an evolution of the C language. The interest in C++ is the concept of Class. A class is a kind of structure that contains variables and functions. A useful concept of oriented languages is the inheritance. If two classes, A and B, inherit from the same class, C. A, B and C will share functions and variables of C. The library managing DICOM files and DICOMDIR files is a good example. DICOMDIR files are a particular kind of DICOM files. DICOMDIR files do not contain any image, but store specific information. The figure 37 shows how the DiLib library is organized.

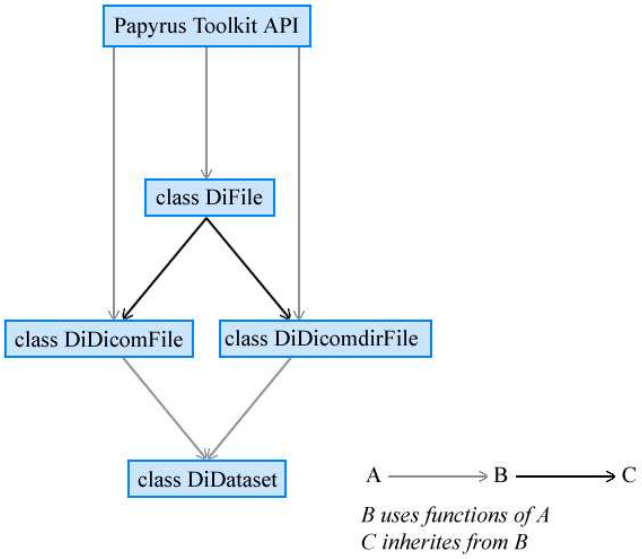


Figure 37. Structure of the DiLib library

The Papyrus toolkit deals with both file formats. It uses exactly the same function for opening and closing files. Records are accessed with exactly the same method. The class DiFile manages functions that the class DiDicomFile shares with the class DiDicomdirFile.

The class DiDicomFile contains specific functions useful to manage images.

Normally the library manages DICOMDIR files, but this class is not complete because I am not able to read the whole file using the Papyrus library. Christian Girard of the Papyrus development [6] team has helped me by e-mail to use his library. His help was useful, but as the reading of this kind of file is not a priority, I have preferred not to spend too much time on it. Between the companion program and the GUI, an agreed protocol has been defined to the DICOM files used by a dataset:

Parameters are sent from the GUI to the X-Ray renderer:

- a string containing the files directory
- A string containing the common part of all DICOM files
- A string containing the digits following and including the first non-zero digit of the first file name of the dataset
- A string containing the digits following and including the first non-zero digit of the last file name of the dataset

This example shows the different parameters sent to describe the dataset, which starts at the file "IM_00005", finishes at the file "IM_00180". Its files are located in the "H:\foot\dicom\":

1st parameter: "H:\foot\dicom\"
2nd parameter: "IM_00"
3rd parameter: "5"
4th parameter: "180"

8.3 CODING STANDARD

As the code of the program will be used by other programmers, it has to be very clear and well documented. Some choice has been done in the method of writing code. The design of the code must be considered for future evolutions of the work to be as easy as possible to read and modify. To increase these two points, some coding standards have been decided.

For example all file names of a library start with "CC" (capital letters for case sensitive file system) if the library is written in C++ or "C" if it is C language. Then there is the first two characters of the library name, as "DI" for a file of the DiLib. It is followed by a '_' and usually by the class name (without the two first characters of the class name and in lowercase). All class names of a library start with the two first characters of the library, as DiDicomFile for the class managing DICOM files. This class definition is written in the "CCDI_dicomfile.h" header file.

It is as important that the variable names are very explicit. For example, g_p_image, is a global variable, its first character is 'g', it is a pointer 'p' to an image. It allows the reader of the code to understand easily by using a standard in the whole code. A class member always starts by an '_', as "_pImage".

8.4 JAVADOC

Programming languages allow comments. The comments can be used by external programs to automatically generate documentation by reading and interpreting comments. Two standards exist. There is the Javadoc style of comments and the QT [20] style. As the javadoc style is probably the most used, I have commented header files using this style.

Using Doxygen [21], an open source program that analyses source files and generates documentation using Javadoc or QT style, I am able to create a documentation in HTML, rich text format (RTF) or man page. Doxygen is class compliant. It determines interactions between classes as inheritance and shows class hierarchy as in Appendix F. It can create documentation about public functions if header files have good comments. It can be very useful for further programmers who would like to use the code because all functions' class methods are listed.

CHAPTER 9 - USERS' TESTS

The subject of this project has been proposed by Philip Cosson. The resulting program should be used by students in radiography. From my point of view as developer, the program has to be tested by the programmer who has created it to reduce the number of bugs, but not exclusively. Final users must test it to give the developer feedback on his work.

Tests have been carried out by different kinds of people. A short "Getting started Guide" has been written. It was given with a tutorial, a test and some printed questions (Appendix G).

9.1 USER'S POINT OF VIEW

9.1.1 *Radiography students*

Each of the two radiography students who have tested the program, have used it for one hour. The setting of a 3D space seems to be very difficult for people who do not use computers a lot. These two students do not often use computers. It is difficult for them to imagine the space in 3D and the purpose of having three different orthographic views. They were, however, able to execute, step by step, the tutorial almost without help. The printed questions were not useful there because there were discussions during the two tests. They have not tried the prepared test because of the duration of our meeting. During those discussions, many points were raised concerning a more user-friendly program such as:

- Distance between objects (as SID[13]) should appear somewhere
- Having the link between colours and objects (e.g. green = patient) inside the interface or in the help menu.
- Having a zoom factor in percentages
- Lock objects together to allow user to move several objects at the same time
- Zooming in on the selected object automatically
- Pan in the perspective view
- A bigger sphere for the X-Ray source

The conclusion of these two tests was that the program is not easy to use but it is possible to learn how to use it. From a developer's point of view it is important to

[13] distance between X-Ray emitter (the source) and receptor (the cassette)

note that the two students do not often use computers; they seemed to be a bit afraid of them; they did not click anywhere to try to understand how it worked. They did not always press the mouse buttons hard enough and were not really familiar with the double-click procedure. Although the tutorial was quite difficult for them, it is impossible to consider that these tests failed, since the discussions are the starting point of possible improvements.

9.1.2 *Students using 3D packages*

Two different CAGTA students with different backgrounds (one 3D programmer and one 3D package user) have tested the GUI. They only needed information about what they have to do with the program. The tutorial and the test were carried out easily and quickly. They were not afraid of clicking anywhere and trying different controls. Since they had no problems using it and had a positive experience with the GUI control, they had some ideas for improving the interface:

- Having a link between colours and objects when the mouse is on an object (like 3DS Max)
- Pan in the perspective view
- A bigger sphere for the X-Ray source

It was interesting to also have the point of view of programmers and 3D package users who do not have the same priorities as radiography students and can understand what they are asking about in terms of programming. Their requests, however, were also the same as those expressed by the future radiographers.

9.1.3 *Computer users without 3D knowledge*

Two other persons have accepted an invitation to test the GUI. One is a musician who plays 3D games quite often and the other one is a network administrator. As contacts were carried out by e-mails, the written questions were useful to guide their criticism.

At the beginning they had problems with the controls, but after “playing” for 20 minutes they were able to understand how it works. The tutorial and the test then seemed to have been easy.

Their needs were totally different. For one of them, the zoom and the movement of the camera in orthographic views were too fast; for the other, the controls were not detailed enough.

9.2 MY OWN POINT OF VIEW

It is a great challenge to do a user-friendly GUI for setting a 3D space. Moreover when users are not familiar with a computer, it becomes harder. The tests done by different kinds of users were successful, because everybody was able to set the 3D space, with less or more time, following the tutorial. Some aspects are very easy to use, such as the selection of an object (except the X-Ray source which is too dark and too small). Others are difficult, such as understanding the different orthographic views.

I agree with all points, which can be changed, but in the timetable I have, I will not be able to change them, I await the opportunity to do them. I think that the first radiography student has reviewed her test quite well; it is difficult to use at the beginning, but it is possible to learn how it works. The 3D gamer has reviewed the GUI differently; it is easy to use when the user understands how the GUI works.

CONCLUSION

The subject of this project has been proposed by Philip Cosson, from the School of Health and Social care at the University of Teesside. He wished to have an X-Ray simulator. As it is a considerable task, it was divided into two projects, a user interface and a X-Ray renderer. Dividing the work in two was sensible, since it was not possible to implement all the details of the GUI. Moreover, it was interesting to work on the same project as someone else, because it was possible to share information. A number of problems were found by one of us and solved by the other or with his help, notably, the case of the image reading from DICOM files.

In this program, a 3D virtual radiography laboratory was created. It is composed of an X-Ray machine (emitter), a cassette (X-Ray receptor) and a patient. The patient is reconstructed in 3D from CT scanned data stored in DICOM files. I have discovered this file format, the current standard of medical image files and I read this kind of file to extract images and some other useful records. The DICOM standard version 3 also deals with another kind of file format, DICOMDIR. I am not able to read it, but one day I might do some more work on this. Before starting the project, It was said that the marching-cube can extract 3D objects from volumetric data. But I had never implemented the algorithm before. For this project, I have successfully written my own version. It creates 3D objects with Gouraud shading from volumetric data obtained by CT scanners. Two meshes are created from the same dataset, a bone model and a skin model. At the beginning, Philip wanted very realistic models with skin texture, but I have found that it was not needed if the material properties (ambient, specular and diffuse colours) are well adjusted. It allows retention of the speed of rendering without the texture mapping.

For the reconstruction, I cannot use the images of the dataset directly, I have to perform some image processing. The problem is the calculation time of image processing algorithms. I have implemented a fast algorithm that removes small artefacts, but it is no longer used because I now use my own method for reducing the level of detail.
To determine the kind of each pixel (soft tissues, bone or air), I just use threshold. The threshold needs user's input. It is not automatic. I have thought about a method for doing it using a histogram, but I did not have time to implement it.

The reconstruction is done with the famous marching-cubes algorithm. It was important for me to implement this algorithm by myself , because next year I am going to become a computer medical imaging student and this algorithm is very famous in 3D medical imaging.

The marching-cube is old (1987), but it is still used for 3D reconstruction of medical volumetric data or the visualisation of mathematical functions in real time using scalar fields. The disadvantage of this method is the high number of vertices and triangles, which compose reconstructed objects. The manipulation of reconstructed objects is impossible when the number of triangles is to high. I must decimate reconstructed polygon meshes. Some algorithms allow reduction of the number of triangles of a 3D object with the loss of only a few details, but as I did not have the time to implement one, I have written my own method, which is faster, because I compress the volumetric data losing accuracy. All voxels are no longer used, in fact, the cube of the marching cube is bigger than a voxel. Moreover, the marching becomes faster. Several hours are needed to reconstruct the skin and bone models using the smallest cube possible, a voxel. The speed is slow because image processing algorithms are usually slow. Moreover, each time that the cube generates a triangle, I have to test if the triangle vertices are already in the list of vertices. If it is the case, I do not add the vertex. Seeking in a link-list is slow. I have speeded up the research by using an sorted linked-list. As I use a sorted double linked-list, I can increase the research by seeking from the beginning of the list if the vertex that I want to add, is nearer the first vertex on the list, or from the end, if it is nearer to the last vertex on the list. Unfortunately, the timetable of the project does not allow me to implement and test if it is actually faster.

The second part of my work is the writing of a GUI. It is tricky to do a user-friendly interface for people who are not in the habit of using computers. The interface has been done in Win32 API. It is the first time that I have written an interface using this API. I have learned, its principle, not without difficulties. It is perhaps often used for writing programs for MS Windows, but it is not well documented. At the end I have an interface, which works and which can be used by radiography students. It is not as user-friendly as I want, but user's feedback came too late to improve much of the GUI. The most important fact, from my point of view, is that the program works and that the students in radiography can use it, probably with some difficulty at the beginning. Two radiography students have tested the interface. They succeeded in using it. They needed a long time to understand the principle of the GUI, because it is not easy to use and set a 3D environment the first time. But I think that the more they have the opportunity to use computers, the faster they can learn how to use the GUI. With a bit more training, they should be able to use it with less problems. My satisfaction comes from the fact that I have learnt how to use Microsoft API and produced an interface, which can be used by radiography students, although more time is needed to implement the requests of the students to make it more user-friendly. This project may serve as a base for a future Virtual Reality project, using other kinds of devices.

I am satisfied with this final project because the final program works. It is not completely finished, but it is perfectly usable. It has been rewarding to get my first GUI using the Microsoft Win32 API. I have enjoyed writing my own implementation of the marching-cube algorithm and getting 3D reconstructed objects of skin and bone using volumetric data stored in standard medical image files. My final project has confirmed my wish of carrying on my study in computer medical imaging research.

REFERENCES

[1]	LORENSEN W E & CLINE H E	*Marching cubes: A high resolution 3D surface construction algorithm,* ACM SIGGRAPH 1987, 163-169
[2]	PIQUET CARNEIRO B et al	*Tetra-cubes: An algorithm to generate 3D isosurfaces based upon tetrahedral,* Anais do IX SIBGRAPI'96, 205-210
[3]	VOLLMER et al	*Improved Laplacian Smoothing Of Noisy Surface Meshes,* EUROGRAPHICS 1999, 131-138
[4]	SCHMEDE W J et al	*Decimation of triangle Meshes,* ACM SIGGRAPH 1992, 65-70
[5]	RAJ SHEKHARLY et al	*Octree-Based Decimation of Marching Cubes Surfaces,* IEEE Visualization 1996, 335-499
[6]	UIN[14] University Hospital Geneva	*Papyrus toolkit,* http://www.expasy.ch/UIN/html1/projects/papyrus/papyrus.html
[7]	CLUNIE D	*Dicom3Tools,* http://www.dclunie.com/dicom3tools.html
[8]	KRUG W & RORDEN C	*ezDicom medical viewer*, www.mricro.com
[9]	UIN[15] University Hospital Geneva	*Osiris,* http://www.expasy.ch/UIN/html1/projects/osiris/osiris.html
[10]	Rubo Medical Imaging BV	*Rubo Medical Imaging,* http://www.rubomedical.com/index.html
[11]	Steinhart Medizinsysteme und elektronische Datenverarbeitung	*Hipax,* http://www.hipax.de
[12]	FIDLER V et al	*Medicview 3D,* http://www.medicimaging.com/medicview/medicview3D.htm
[13]	University of Umea	*Virtual Radiography,* http://www.vrlab.umu.se/forskning/vradiography_eng.shtml
[14]	REMS Software	*SimXray,* http://www.simxray.com
[15]	MONTES M et al	*Virtual radiation laboratory,* http://www-personal.engin.umich.edu/~godfroy/Radiation/hazard.html
		Team HAZARD

[14] Unité d'Imagerie Numérique (Digital Imaging Unit)

[15] Unité d'Imagerie Numérique (Digital Imaging Unit)

[16] BOURKE P *Polygonising a scalar field,*
http://astronomy.swin.edu.au/~pbourke/modelling/polygonise/
Swinburne University of Technology (Australia), 1997

[17] BLOYD C *Marching example program*
http://astronomy.swin.edu.au/~pbourke/modelling/polygonise/marchingsource.cpp

[18] BLOYD C *Roentgen And The Discovery Of X-rays*
http://www.xray.hmc.psu.edu/rci/ss1/ss1_2.html

[19] MEUNIER J http://www2.iro.umontreal.ca/~meunier

[20] TROLLTECH *QT*
http://www.trolltech.com/

[21] DOXYGEN *Doxygen*
http://www.doxygen.org /

[22] Right Hemisphere *3D Exploration*
http://www.righthemisphere.com

[23] Avid *SoftImage*
http://www.softimage.com

[23] Viewpoint corporation *Internet browser plugins*
http://www.viewpoint.com

[24] MALLARD D *The body scanner*
http://www.technologyscotland.org/pioneering/body_body.html

[25] TEECE D *Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy*
http://www.lib.uconn.edu/Economics/Teece RP (1986).pdf

[26] DISCREET *3DS Max*
http://www.discreet.com

[27] ADOBE *Photoshop*
http://www.adobe.com

APPENDIX A – DICOM file extract

```
0002,0000,File Meta Elements Group Len: 164
0002,0001,File Meta Info Version: 256
0002,0002,Media Storage SOP Class UID: 1.2.840.10008.5.1.4.1.1.2.
0002,0003,Media Storage SOP Inst UID:
1.2.840.113619.2.30.1.1762369857.2100.1020760049.575
0002,0010,Transfer Syntax UID: 1.2.840.10008.1.2.1.
0002,0012,Implementation Class UID: 1.3.46.670589.5.2.13
0008,0000,Identifying Group Length: 732
0008,0001,Length to End: 531856
0008,0005,Specific Character Set: ISO.IR 100
0008,0008,Image Type: ORIGINAL\PRIMARY\AXIAL
0008,0016,SOP Class UID: 1.2.840.10008.5.1.4.1.1.2.
0008,0018,SOP Instance UID:
1.2.840.113619.2.30.1.1762369857.2100.1020760049.575
0008,0020,Study Date: 20020507
0008,0021,Series Date: 20020507
0008,0022,Acquisition Date: 20020507
0008,0023,Image Date: 20020507
0008,0030,Study Time: 184956.000000
0008,0031,Series Time: 185459.000000
0008,0032,Acquisition Time: 185521.000000
0008,0033,Image Time: 185842.000000
0008,0050,Accession Number:
0008,0060,Modality: CT
0008,0070,Manufacturer: GE MEDICAL SYSTEMS
0008,0080,Institution Name: NEUROSCIENCES  CENTRE  NGH
0008,0090,Referring Physician's Name:
0008,1010,Station Name: OC01.OC0
0008,1030,Study Description: FOOT
0008,103E,Series Description: 3/3 KNEES
0008,1070,Operator's Name: KM
0008,1090,Manufacturer's Model Name: HiSpeed CT/i
```

APPENDIX B - Image processing

These pictures come from the Jean Meunier's web site at the University of Montréal [19]

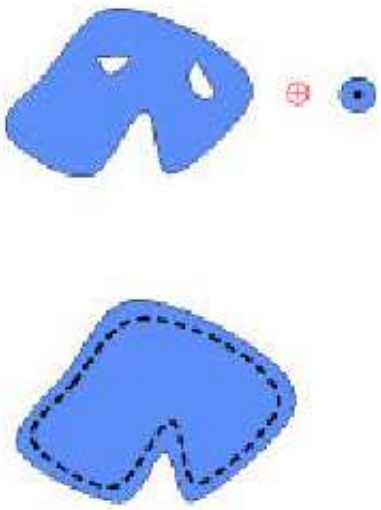


Figure 38. Dilatation

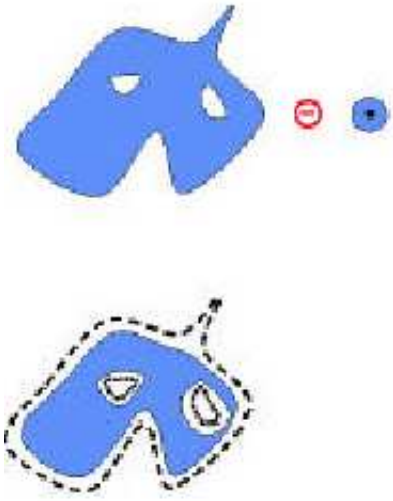


Figure 39. Erosion

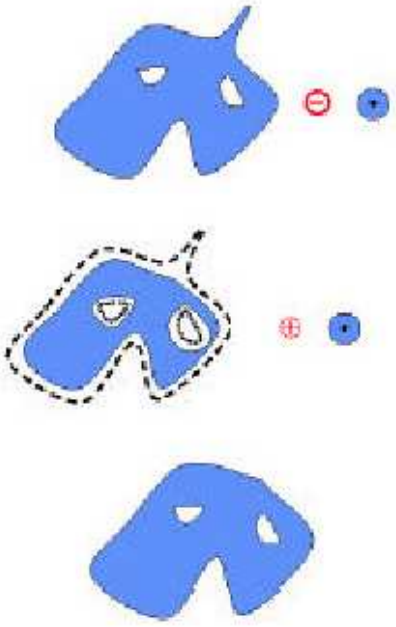


Figure 40. Open

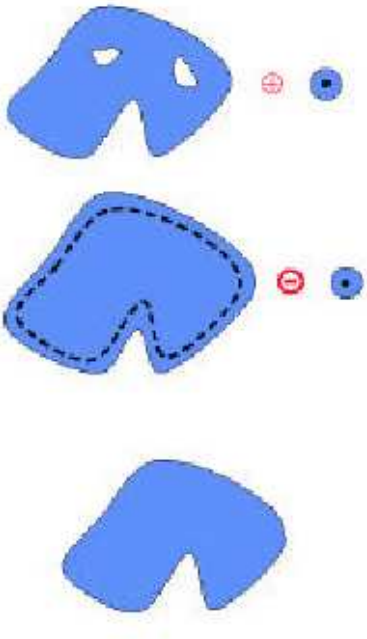


Figure 41. Close

APPENDIX C – Windows

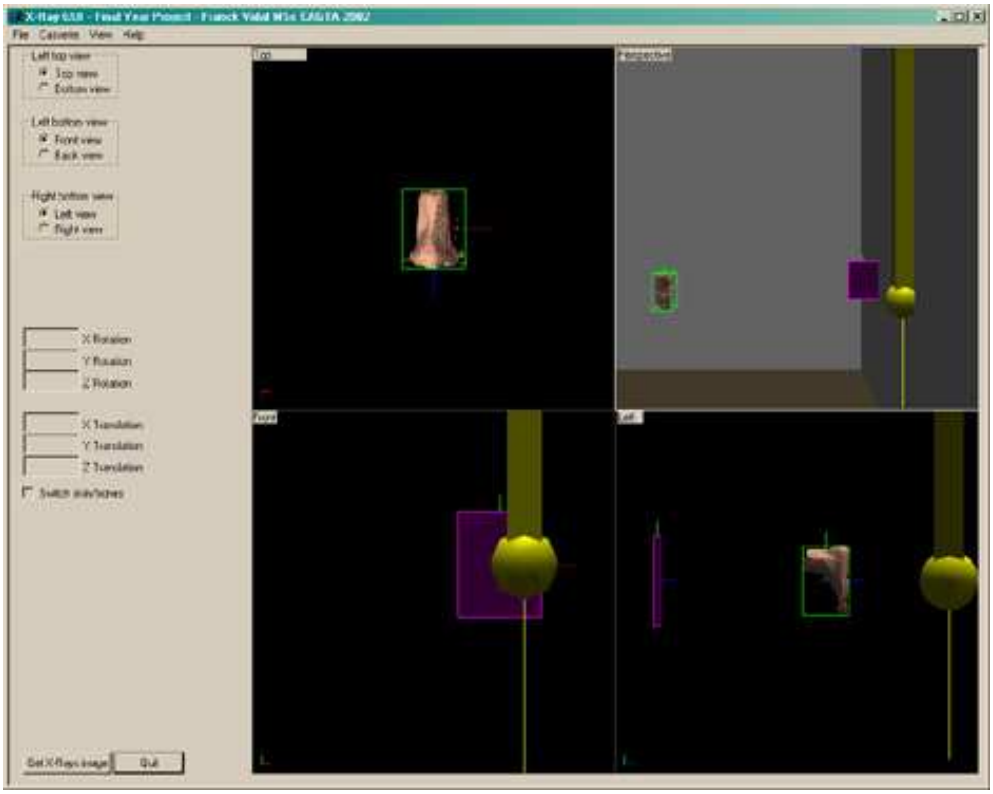


Figure 42. The main window of the GUI

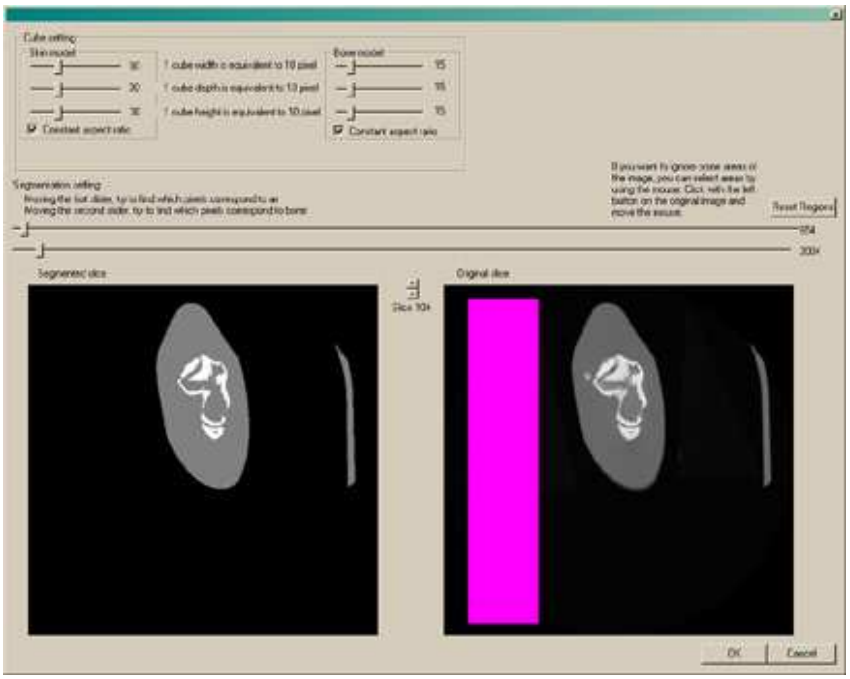


Figure 43. Window used for setting the reconstruction

APPENDIX D – X-Ray GUI file format

All values designed as ‘code’ are constant values inserted in files for if X-Ray GUI files are corrupted or not.

An unsigned char, the file version

An unsigned char, the patient code
An unsigned short, the patient id

An unsigned char, the study code
An unsigned short, the study id

An unsigned char, the series code
An unsigned short, the series id

An unsigned char, the bounding box code
An short, the minimum value along the x-axis
An short, the maximum value along the x-axis
An short, the minimum value along the y-axis
An short, the maximum value along the y-axis
An short, the minimum value along the z-axis
An short, the maximum value along the z-axis

An unsigned char, the skin object code
An unsigned int, the number of vertices of the skin object
An unsigned int, the number of triangles of the skin object
An short, the minimum value along the x-axis
An short, the maximum value along the x-axis
An short, the minimum value along the y-axis
An short, the maximum value along the y-axis
An short, the minimum value along the z-axis
An short, the maximum value along the z-axis
An unsigned char, the skin object vertices code
Floats, the vertices of the skin object (there are three floats per vertex)
An unsigned char, the skin object index code

Unsigned ints, the indexes of the skin object (there are three unsigned ints per index)
An unsigned char, the bone object normal code
Floats, the vertices of the bone object (there are three floats per normal)

An unsigned char, the bone object code
An unsigned int, the number of vertices of the bone object
An unsigned int, the number of triangles of the bone object
An short, the minimum value along the x-axis
An short, the maximum value along the x-axis
An short, the minimum value along the y-axis
An short, the maximum value along the y-axis
An short, the minimum value along the z-axis
An short, the maximum value along the z-axis
An unsigned char, the bone object vertices code
Floats, the vertices of the bone object (there are three floats per vertex)
An unsigned char, the bone object index code
Unsigned ints, the indexes of the bone object (there are three unsigned ints per index)
An unsigned char, the bone object normal code
Floats, the vertices of the bone object (there are three floats per normal)

An unsigned char, the end of file code

APPENDIX E – Use of the command line arguments

The following information is needed by the rendering program. This order is agreed by Daniel Deprez and me. 0a, 0b, 0c and 0d is a temporary solution until we are able to read DICOMDIR file information.

0a dataset directory, a string (e.g. "H:/foot/dicom/")
0b filename const, a string (e.g. "IM_00" is common to IM_00005 and IM_00180)
0c filename var first, a string (e.g. 5 - all digits following and including the first non-zero digit in IM_00005)
0d filename var last, a string (e.g. 5 - all digits following and including the first non-zero digit in IM_00005)

1 Patient ID, an unsigned short;
2 Study ID, an unsigned short;
3 Series ID, an unsigned short;

4 X component of the X-Ray source position, a float;
5 Y component of the X-Ray source position, a float;
6 Z component of the X-Ray source position, a float;
7 Rotation angle of the X-Ray source throw the X-axis, a float;
8 Rotation angle of the X-Ray source throw the Y-axis, a float;
9 Rotation angle of the X-Ray source throw the Z-axis, a float;

10 X component of the patient position, a float;
11 Y component of the patient position, a float;
12 Z component of the patient position, a float;
13 Rotation angle of the patient throw the X-axis, a float;
14 Rotation angle of the patient throw the Y-axis, a float;
15 Rotation angle of the patient throw the Z-axis, a float;

16 Width of the cassette, an unsigned short;
17 Height of the cassette, an unsigned short;
18 Depth of the cassette, an unsigned short;
19 X component of the cassette, a float;
20 Y component of the cassette, a float;
21 Z component of the cassette, a float;

22 Rotation angle of the cassette throw the X-axis, a float;
23 Rotation angle of the cassette throw the Y-axis, a float;
24 Rotation angle of the cassette throw the Z-axis, a float;

Before sending the parameters, the program must create 25 strings. The strings size cannot exceed 255 characters (256 with the character of end of string '\0'). The original kind of the parameters can be float, unsigned short or string. As parameters are different kinds, the program converts variables into a string using the "sprintf" standard C function of the "stdio" library.

The GUI can then create and execute the new child process using the "spawnl" function. It is not a POSIX[16] function. It builds an array of strings compatible with the "*argv*[ ]" array (argument of the main function of a C/C++ program) before calling the "spawn" function. The "spawn" function is part of the POSIX standard. Another family function can create a new child process with command line arguments; it is the "exec" function. When a call to an "exec" function is successful, the new process is placed in the memory previously occupied by the calling process. As the calling process has no more memory, it is not possible to come back to the GUI after launching the X-Ray renderer. Spawn like functions create a new child process, which does not use the memory of the calling process, but uses its own.

This code shows the different step of the creation of a new child process:

```
// Creation of 25 strings
char dataset_file_name[MAX_STRING_LENGHT]; // string

char patient_id[MAX_STRING_LENGHT]; // USHORT
char study_id[MAX_STRING_LENGHT]; // USHORT
char serie_id[MAX_STRING_LENGHT]; // USHORT

.
.
.

// Set up the 25 strings using "sprintf"
sprintf(dataset_file_name, g_p_interface->_Dataset.GetFileName());

sprintf(patient_id, "%i", g_p_interface->_Dataset.GetPatient());
sprintf(study_id, "%i", g_p_interface->_Dataset.GetStudy());
sprintf(serie_id, "%i", g_p_interface->_Dataset.GetSerie());

// Creation of a new child process
int error_code(spawnl(_P_NOWAIT, /* The calling process does not wait
                                 * the end of the child process.
                                 * The two processes are
                                 * asynchronous
```

[16] Portable Operating System Interface based on UniX

```
            */
child_process_file_name, // File name
dataset_file_name,
patient_id,
study_id,
.
.
.
```

APPENDIX F – Javadoc

Documentation generated showing the class hierarchy in the DiLib library

DiLib Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- **DiDataset**
- **DiFile**
  - **DiDicomdirFile**
  - **DiDicomFile**
- **DiRoi**

Documentation generated showing the inheritance for the DiFile class. The classes DiDicomdirFile and DicomFile inherit of the virtual calss DiFile.

Inheritance diagram for DiFile:

Documentation generated showing the description of the Open method of the class DiFile.

virtual short DiFile.Open (char * *aFileName* = 0) [virtual]

---------- Open a dicom file

Parameters:

a string: the name of the dicom file

Returns:

a short: the file number reference or an error code ----------

Reimplemented in **DiDicomFile**.

The following code has been used to generate the previous documentation of the Open method of the class DiFile. It starts by /** as the javadoc style. @param and @return are interpreted to know the parameters of the method and its returned value.

```
/**-------------------------------------------------------------
 *    Open a dicom file
**
*     @param      a string:   the name of the dicom file
*     @return a short:  the file number reference or an error -*
      code
* -------------------------------------------------------------*/
virtual inline short Open(const BaString& aFileName)
{
      _FileName = aFileName;

      return Open();
}
```

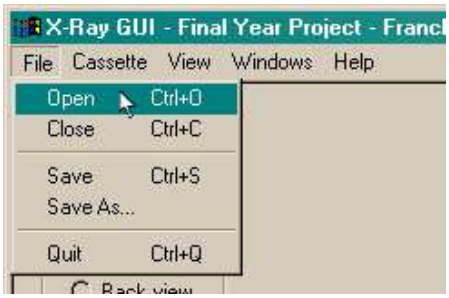
APPENDIX G – Test support materials

Getting started Guide

1. Launch the program and open a dataset

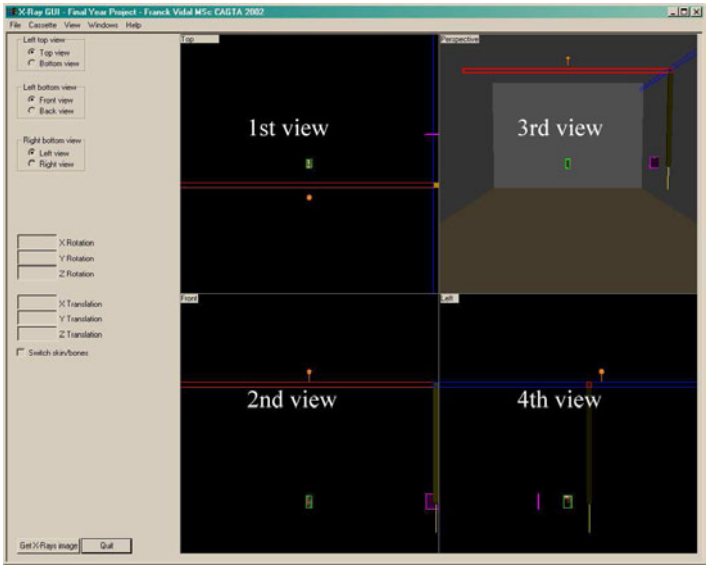
- In the home directory of X-Ray GUI program, there is a file called "XRayGUI.exe". Double clicking on it will launch the X-Ray GUI program.

- Choose the dataset by using the keyboard shortcut Ctrl+O or using the "File->Open" menu.
Then, choose the file containing the objects reconstructed from CT-scanned data "foot_s5_b5.xrg".



2. Generalities

- When the file is loaded, you are able to see this window.



- Double-click with the left button on a view to fill the window with it.

- Double-click again to restore 4 views in the window.

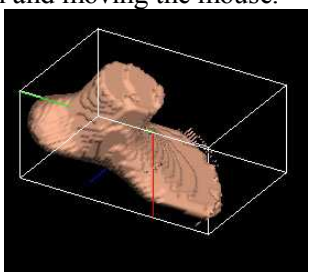
- Select a cassette size using the "Cassette" menu.

- The cassette bounding box is purple, the patient bounding box is green and the X-ray source is red, blue and yellow. A selected object is white.

- Zoom in/out by pressing the 'x' key and by pressing the left button of the mouse and moving the mouse up and down.

- Restore the default view states using the “View” menu.

- Rotate selected object by pressing the “R” key, pressing a mouse button and moving the mouse.
The left button corresponds to a rotation through the local x-axis of the object (red line).
The middle button corresponds to a rotation through its y-axis (green line).
The right button corresponds to a rotation through its z-axis (blue line).



3. Top/Bottom, Front/Back and Left/Right views

- Move the camera by pressing the ‘x’ key and by pressing the right button of the mouse and moving the mouse.

- Select an object in a view by clicking with the left button on this object.

- Move any selected object by pressing the right button of the mouse and moving the mouse.

- Rotate any selected object by pressing the middle button of the mouse and moving the mouse.

4. Perspective view

- The selection of an object is a loop:
No object -> Patient -> Cassette -> X-Ray emitter -> No object
You can change of selected object with a double right-click.

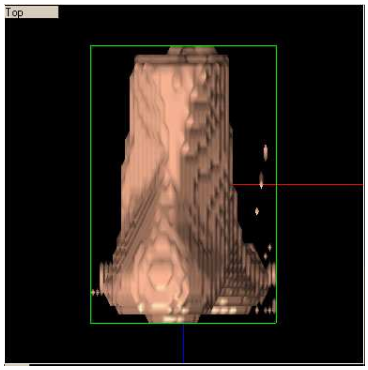
- Move up/down/left/right any selected object by pressing the “T” key, pressing the left button and moving the mouse.

- Move forward and backward selected object by pressing the “T” key, pressing the right button and moving the mouse.

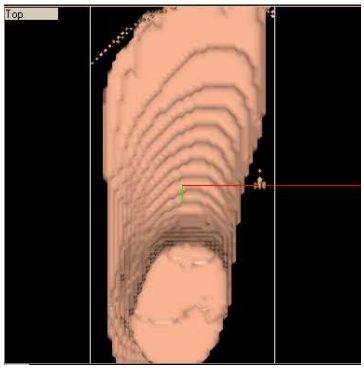
Tutorial

- Load the “foot_s5_b5.xrg” file.

Zoom in the top view (Fill the view with the foot).

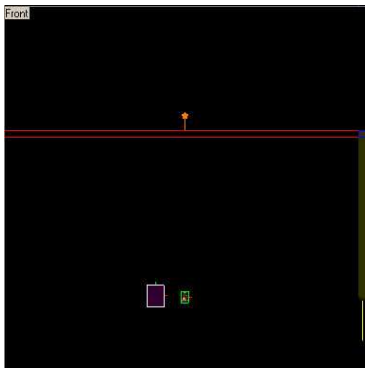


Press and hold the ‘R’ key and use left mouse button to rotate the foot around the x-axis(its red line). Rotate through 90 degrees (until the top of the foot is visible).

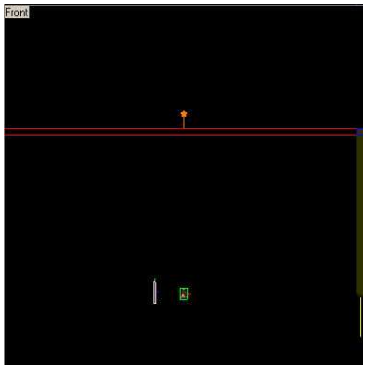


Zoom out the top view (Fill the view with 3 objects (cassette, patient and X-Ray source)) by using “View->Reset->Left Top” menu. This action restores the default view setting.

In the front view, select the cassette with the mouse.
Move it to the left of the foot.

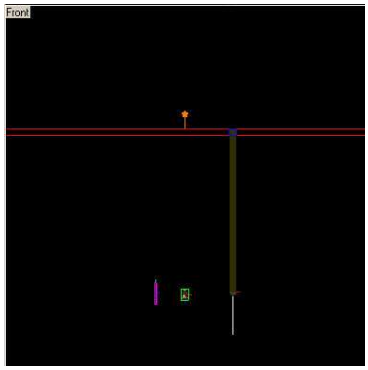


Press and hold the ‘R’ key and use left mouse button to rotate the cassette around the y-axis(its green line). Rotate through 90 degrees (until the top of the foot is visible).

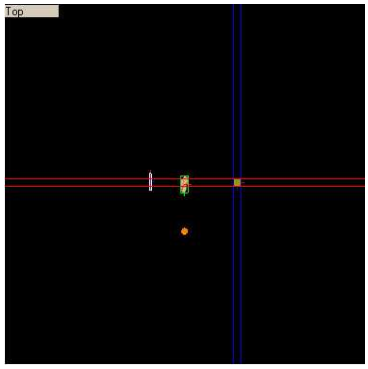


Rotate the cassette with an angle of 90 degrees through its y-axis (Press the ‘R’ key. Keep it down, Press the middle mouse button and move the mouse).

Select the X-Ray source with the mouse.
Move it to the right of the foot.

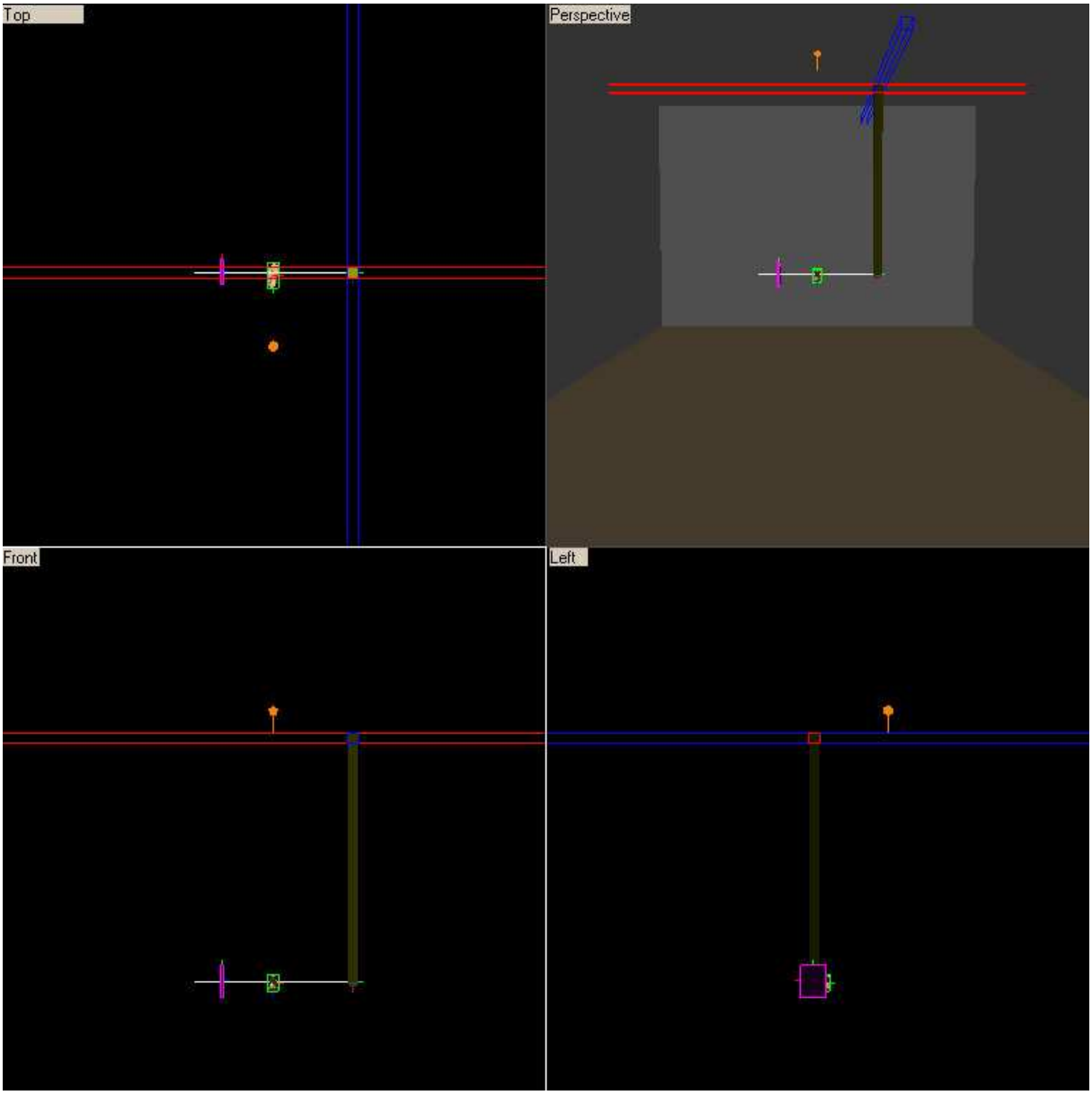


In the top view, align the cassette and the X-Ray emitter with the foot.



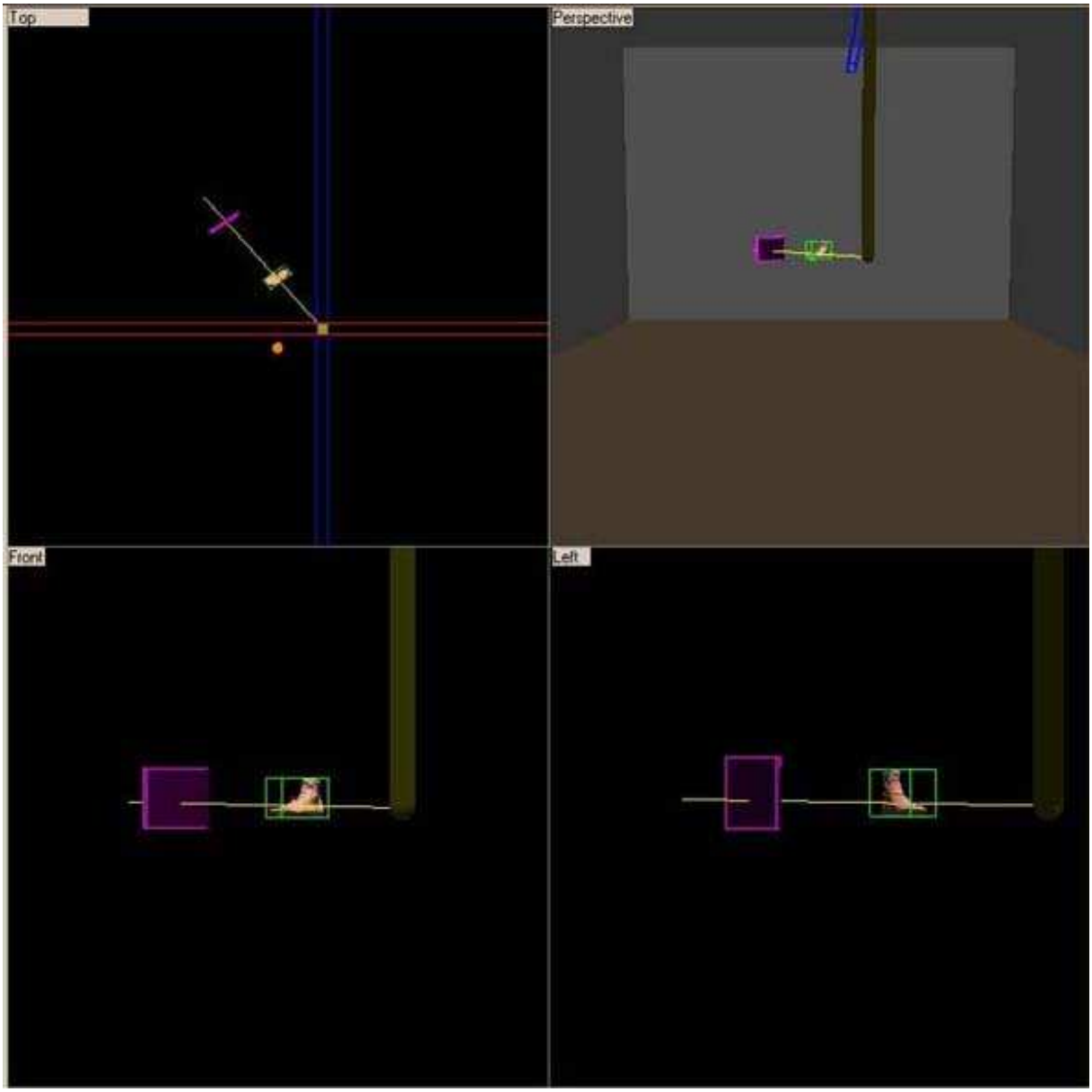
In the left view, Rotate the X-Ray emitter with an angle of -90 degrees through its z-axis (blue line).
Press the ‘+’ key to see if the X-Ray beams intersect the cassette and the foot.

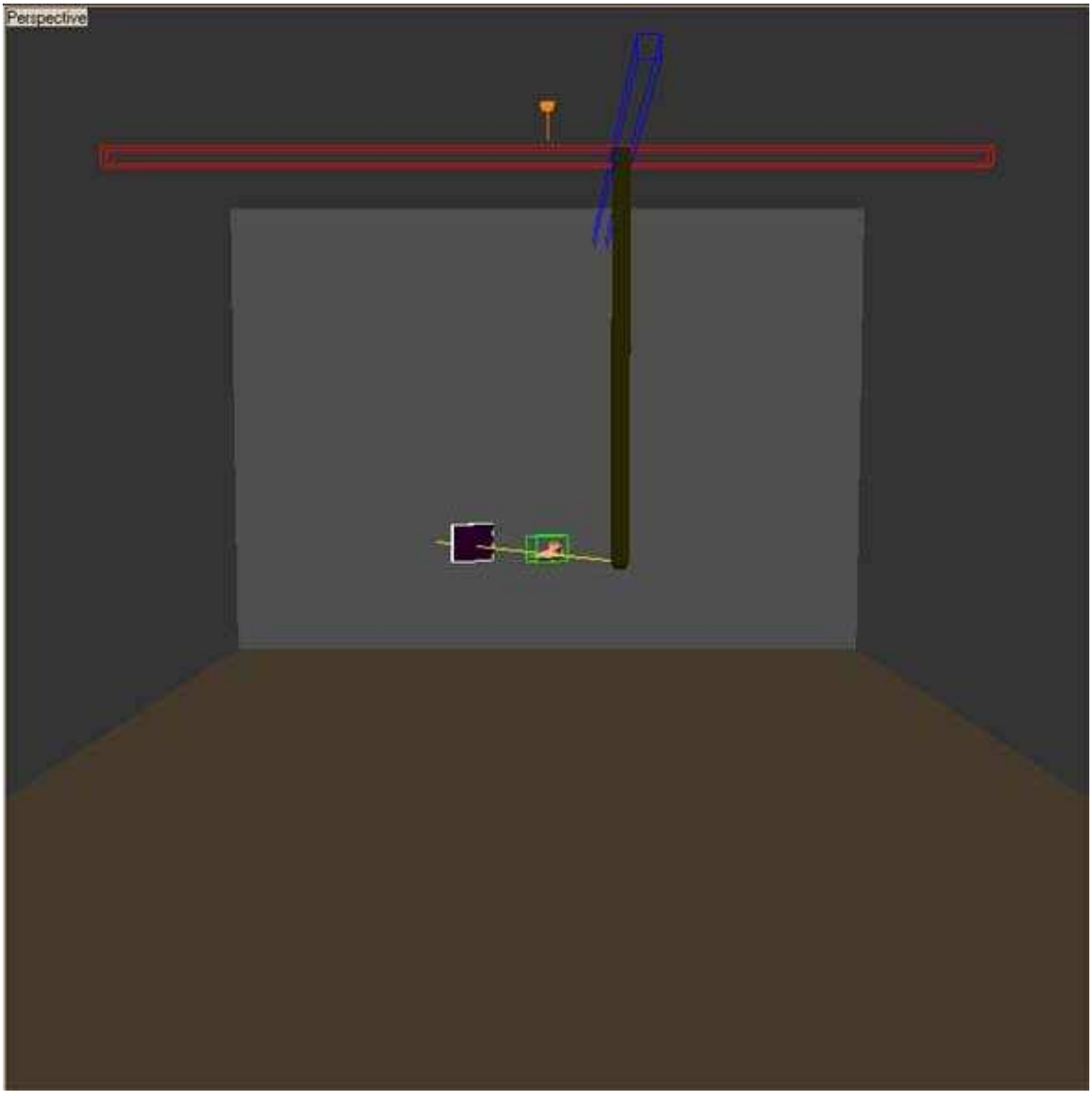
You should get this final result:



Test

Try to get the same result. For the foot, only rotations (no translations).




Perspective

12 questions

0 Cannot be worse

1 Bad / Bad idea / Difficult to use

2 Quite good / Quite good idea / Quite easy to use

3 Good / Good idea / Easy to use

4 Very good idea / Very user-friendly

		0	1	2	3	4
1	The selection of objects in the orthographic views (top/bottom/front/back/left/right views)					
2	The selection of objects in the perspective view					
3	The translation of objects in the orthographic views (top/bottom/front/back/left/right views)					
4	The translation of objects in the perspective view					
5	The rotation of objects in the orthographic views (using the only the middle mouse button) (top/bottom/front/back/left/right views)					
6	The rotation of objects in the orthographic views (using the keyboard shortcut 'R' and one mouse button) (top/bottom/front/back/left/right views)					
7	The rotation of objects with the link between mouse button and axis (using the keyboard shortcut 'R' and one mouse button)					
8	Zoom in/out, is it easy					
8	Zoom in/out, is it useful					
9	The idea of the change of the size of the line, which indicates the X-Ray direction, to see if the X-Ray beams intersect or not the patient and the cassette					
10	The link between objects colour and the kind of objects (e.g. green bounding box for the patient, purple for the cassette, white for selected object)					
11	The use of bounding box, is it useful					
12	In the perspective view, is the room (walls, ceiling and ground) helpful to see relative position of objects?					

Any comments:

What can be done to make a better program?