UCLouvain

École polytechnique de Louvain

Interoperable encoding and 3D printing of anatomical structures resulting from manual or automated delineation

Author: **Thibault Gregoir**
Supervisor: **Sébastien Jodogne**
Readers: **Benoît Macq, Greet Kerckhofs**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Acknowledgments

First and foremost, I would like to express my gratitude to Professor Sébastien Jodogne for initiating this thesis. He has provided me with the invaluable opportunity to work on a subject that encompasses various aspects of computer science, including algorithmic, software engineering, and artificial intelligence. This amalgamation of disciplines has allowed me to experience a sense of novelty, as each component of the thesis requires distinct and diverse approaches. I would also like to extend my thanks to Professor Jodogne and his assistant, Amaury Fierens, for their availability, guidance, advice, and feedback throughout this year.

Additionally, I would like to acknowledge the unwavering support of my family and friends throughout this year and my entire academic journey. I am especially grateful to my friend Théo and my grandmother, Béatrice, who served as external readers, contributing to the realization of this project. Their involvement and assistance have been invaluable.

Thirdly, I would like to express my gratitude to the Makilab of Louvain-La-Neuve for their warm welcome and the assistance they provided me in utilizing and printing the various models resulting from this master's thesis.

Furthermore, I would like to notice the use of ChatGPT in this work as authorized by UCLouvain. I utilized this tool as a source of inspiration for certain sections and to correct bugs and errors in both code and text. It is important to note that ChatGPT was not employed to create or complete the work on my behalf, but rather to streamline certain aspects of the project, saving time and enhancing efficiency in my place.

Abstract

The understanding and visualization of the human body have been instrumental in the progress of medical science. Over time, the shift from cumbersome and invasive methods to modern scanners highlights the significance of expertise in retrieving, utilizing, and comprehending the resulting data. 3D rendering and printing of organic structures offer promising applications such as surgical planning and medical education. However, challenges arise as technological advancements generate increasingly vast amounts of data, necessitating seamless manipulation and transfer within the medical field. The goal of this master thesis is to explore interoperability in encoding 3D models and the ability to print those models resulting from 3D reconstruction on medical input data. This exploration will be done for models that were originally segmented by manual delineation or in an automated way. Different parts of this thematic were already explored in a specific way like for the surface reconstruction or the automatic segmentation. The idea here will be to combine the different aspects of this thesis in a single tool available and usable by everyone.

Contents

Introduction **1**

1 3D reconstruction in medical field : State of the art **3**
1.1 3D models Reconstruction . . . 3
1.2 Algorithms for surface reconstruction . . . 4
1.2.1 Reconstruction from images . . . 4
1.2.2 Survey of algorithms . . . 4
1.2.3 Reconstruction from point cloud . . . 4
1.2.3.1 Reconstruction from contours . . . 5
1.3 Specific method for surface reconstruction . . . 6
1.3.1 Poisson surface reconstruction algorithm . . . 6
1.3.2 Ball pivoting algorithm . . . 6
1.3.3 Marching cubes algorithm . . . 6
1.3.3.1 Marching cubes algorithm with interpolation . . . 8
1.4 Files format . . . 8
1.4.1 Output formats . . . 8
1.4.2 Input Formats . . . 10
1.5 Techniques of Segmentation . . . 12
1.5.1 Algorithm Methods . . . 12
1.5.2 AI Method . . . 14

2 Problem statement and objectives of the master thesis **17**

3 Methodology **18**
3.1 First look at the data . . . 18
3.2 Choice of the algorithm . . . 19
3.3 Implementation of the Marching Cubes Algorithm . . . 20
3.4 Improve the results of Marching Cube algorithm . . . 21
3.4.1 Marching cubes with interpolation . . . 21
3.4.1.1 Preprocessing . . . 21
3.4.1.2 Modification on the algorithm . . . 22
3.4.2 Post-processing . . . 22
3.5 Creation of DICOM STL Encapsulated . . . 23
3.6 Creation of a webViewer . . . 24
3.6.1 Schema of the process and presentation of the tools . . . 24
3.6.2 Parsing the DICOM files and Retrieving of STL files . . . 25
3.6.3 Creating a scene and loading STL files . . . 26
3.6.4 Positioning the 3D models . . . 26
3.7 Print the models . . . 27
3.8 Use of AI . . . 28
3.9 User Interface . . . 29

4 Results **30**
4.1 Results of Marching Cubes algorithm . . . 30
4.2 Improved Results of the Marching Cubes Algorithm . . . 34
4.2.1 Results for Marching Cubes with interpolation . . . 34
4.2.2 Results for post-processing the meshes . . . 36

4.2.3 Results by combining the two methods . . . 37
4.3 Creation of DICOM STL Encapsulated . . . 38
4.4 Use of AI . . . 41
4.5 Web Viewer . . . 43
4.6 Printed Results . . . 43
4.6.1 Problems . . . 46
4.7 User Interface . . . 48

Conclusion **51**
Contributions . . . 51
Discussion . . . 51
Future Works . . . 53

Bibliography **54**

A Comparison between the original models and the smoothed version **58**

B Comparison between the printed models and their corresponding digital version **60**
B.1 LUNG_R Model . . . 60
B.2 LUNG_L Model . . . 62

Introduction

Over the years, the understanding and visualization of the human body have played a crucial role in the advancement of medicine. In the past, cumbersome and invasive methods were employed. Fortunately, the use of modern scanners has greatly simplified these techniques, although they still require extensive expertise to fully leverage the technologies and achieve optimal results. Accurate acquisition and visual representation of anatomical structures remain essential for precise diagnostics, optimal surgical planning, and quality medical education.

However, with technological advancements, a significant amount of data is generated. It has become paramount to easily manipulate and transfer this data within the medical field to facilitate patient monitoring. From a technical point of view, 3D rendering and 3D printing of organic structure have the potential to help healthcare professionals in their daily work. These technologies can lead to a better description of the disease or a better visualization of the problems. It can also help with better surgical preparation through specific training for the patient's body or the ability to create custom implants tailored to the unique needs of each situation.

The potential of 3D imagery and 3D prints is crucial in enhancing the patient's experience. Over time, erosion of trust in authority figures such as scientists and medical professionals, coupled with the proliferation of misinformation online, has led to increased apprehension in the field of medicine and healthcare. Additionally, fear of surgery and the challenge of comprehending medical procedures can contribute to this unease surrounding healthcare.

However, by providing patients with improved visualization and a deeper understanding of their own body, including the ability to manipulate 3D prints of their organic structure, can greatly assist in alleviating their concerns and fostering acceptance of their conditions. This approach offers a non-invasive means of visualizing data without heavy procedures. By utilizing 3D imagery and prints, patients can have a more comprehensive view of their medical situation, enabling better follow-up and facilitating more informed discussions with their healthcare providers. This technology empowers patients to actively engage in their own healthcare journey and enhances their ability to comprehend the intricacies of treatments and interventions. Ultimately, it contributes to building trust, improving patient's satisfaction, and promoting overall well-being.

A remaining problem is that it could take a huge amount of time for the segmentation and the print of 3D models. But a least one of those points could be simplify by automatic detection/segmentation.

Automatic detection in medical imaging is an essential tool for clinicians, radiologists, and researchers. It helps to identify abnormalities, track changes over time, and make accurate diagnoses. The use of artificial intelligence (AI) in medical imaging has revolutionized the field, providing improved accuracy and efficiency in detecting and diagnosing diseases. AI can analyse vast amounts of data quickly and accurately detecting subtle changes that may not be visible to the human eye. The utilization of Artificial Intelligence in tasks such as image segmentation or automated detection of organic structures has the potential to significantly save time for technicians and radiologists. With AI, these tasks that would typically take hours for a human can be accomplished within minutes. This remarkable acceleration in processing time can streamline workflows and expedite diagnoses, allowing medical professionals to allocate their valuable time more efficiently.

The various aspects mentioned earlier are sometimes utilized in certain aspects of treatments or software. However, the true potential lies in combining all these technologies into a unified, user-friendly platform that can really help the medical field.

This master's thesis will delve into various aspects and outcomes of 3D reconstruction from medical data to achieve the different goals expected for this work. Following an introductory chapter that describes the current state of the art in surface reconstruction and provides important background knowledge, the methodology chapter will outline the approach to be employed in addressing the problem statement of this thesis. The results chapter, along with the subsequent discussion, will enable a comprehensive understanding and demonstrate the potential impact of this thesis on patients and healthcare professionals.

Chapter 1

3D reconstruction in medical field : State of the art

1.1 3D models Reconstruction

Reconstructing 3D models from various data has been a long-standing challenge that scientists have been trying to solve for many years. This problem has applications in fields such as case studies to enhance judgment or upgrading special effects in movies, extending beyond the healthcare industry. However, the difficulty lies in determining the expected results while considering various constraints in order to choose the most suitable method to create 3D representation of an object.

When dealing with 3D models or data, reconstruction should not be the sole focus. Many factors influence the algorithms and the approach to problem-solving. For instance, it is important to consider whether the 3D models need to be created independently or in relation to other elements. Additional information such as lighting, the nature of textures, or the model's interaction with the environment, could also need to be incorporated. While some of these details may seem insignificant, they can greatly impact the accuracy of the final result.

Algorithms used for reconstruction can be computationally demanding. Depending on the desired outcome, methods should be configurable to accommodate different requirements. For approximations or rough representations, fast-computing models might do the trick. However, for a more visually appealing effect, a significant amount of time may be necessary to achieve the desired level of detail and accuracy.

When considering criteria for choosing a 3D reconstruction algorithm, the completeness of the resulting model is also a crucial factor. While some algorithms can generate models from point clouds, the results may not always be complete, with holes or rough patches in some parts. The Advancing Front Surface Reconstruction algorithm [1] and the Polygonal Surface Reconstruction algorithm [2] included in the Computational Geometry Algorithms Library (CGAL) are examples of this kind of method. To address this issue, post-processing techniques can be applied to the mesh, or more points can be added to the input to reduce the risk of incomplete results.

Methods can make the assumption of well collected data but inaccuracies in the sampling process are likely and can lead to reconstruction problems. For example, holes, non-homogeneous densities may be present. Errors or noise can also affect the data. Therefore, pre-processing steps are sometimes crucial in the reconstruction of 3D models.

The following section will provide information and methods specifically related to surface reconstruction. It will bring a brief overview on some algorithms while going deeper in others. Additional information regarding the medical field, saving models, and automatic segmentation will also be explained to provide a comprehensive context for this work.

1.2 Algorithms for surface reconstruction

1.2.1 Reconstruction from images

A first way to reconstruct models is by using different types of images such as photos, scans[3], and even video frames. Photogrammetry and stereovision are just two examples of such methods, which share the common idea of using algorithms to reconstruct 3D models from one[4, 5] or more images[6–10]. These algorithms, sometimes enhanced by artificial intelligence [4, 11], use the information present in the images to determine depth and shape from 2D data.

One of the advantages of these methods is that additional information, such as colors and material structures, can be easily obtained and incorporated into the 3D models, resulting in more realistic and detailed models. Additionally, the acquisition of images is becoming increasingly easier, thanks to everyday objects equipped with cameras such as smartphones and drones[12]. The possibilities for objects that can be reconstructed are also extensive, ranging from landscapes and buildings to small objects.

1.2.2 Survey of algorithms

Surface reconstruction is a long-standing challenge in computer science, and surveys have been developed to classify the methods used to solve this problem[13–15]. Many surveys exist that provide an overview of the problem and a showcase for various algorithms, almost like a catalog. Surveys can be useful because they present a lot of information in one place and allow multiple algorithms to be compared simultaneously.

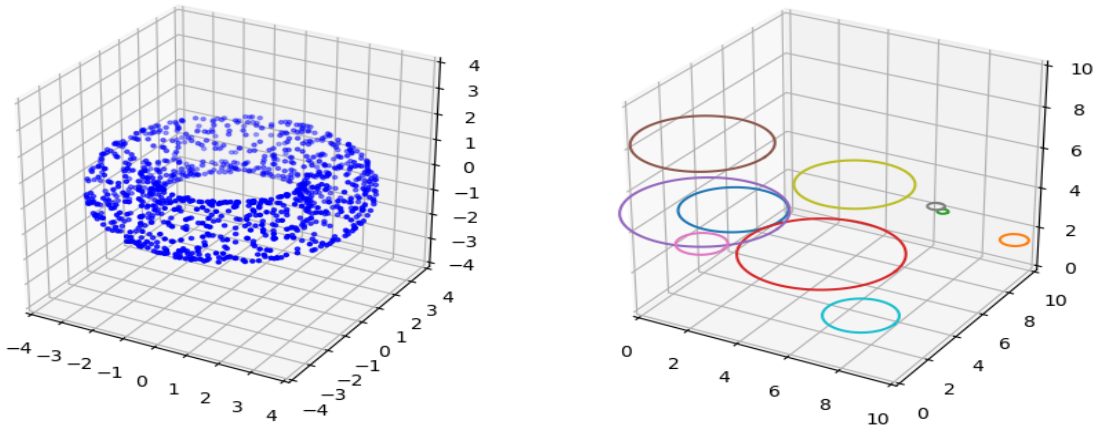
Comparison between different reconstructing methods has been tried in surveys. However, they lack a fair base of comparison since models sometimes require different inputs or produce dissimilar outputs. There is often a trade-off between time-consuming computations and the level of detail in the output. Additionally, authors can make errors in the way they implement or test algorithms. For example, in the paper "Survey on 3D Surface Reconstruction"[16], the authors did not report the number of triangles generated by the marching cubes algorithm, which is concerning since the algorithm generates triangles in a well-defined way.

While surveys can provide a helpful introduction to the field of surface reconstruction, in-depth research is necessary to identify the most efficient methods for solving specific problems. It is important to combine the information obtained from surveys with other evaluation methods, such as experimental performance tests or more detailed assessments of the quality of the surface reconstruction output.

1.2.3 Reconstruction from point cloud

Cloud point is usually a type of data obtained during a surface reconstruction process. This data can originate from the original dataset or be derived through data processing. Developing methods that can effectively handle various types of point clouds can be challenging. The utilization of mathematical techniques becomes crucial, as the presence of randomly distributed points makes it difficult to generalize the reconstruction process[17, 18]. As mentioned previously, it is important to note that point clouds are not always homogeneous and may contain noise. In such cases, Random Sample Consensus (RANSAC)[19] is often employed to determine parameters that can be utilized by algorithms for accurate surface reconstruction.

For the reconstruction process itself, there are numerous options that can be applied to the point cloud data. One such option is utilizing Delaunay triangulation[17], which can be beneficial on its own or in combination with other approaches. Researchers have also explored the use of weighted minimum surface energy, aiming to minimize it in order to reconstruct a model[20]. Another approach involves identifying the general shape through the point cloud[21]. These are just a few examples, but they demonstrate the wide range of possibilities available to generate a 3D model as the output.



(a) Example cloud point representing a torus (b) Example contours representing an abstract form

Figure 1.1: Example of input data

1.2.3.1 Reconstruction from contours

Contours are a special case of point clouds where the points lie on specific values on the X, Y, or Z axis depending on the dataset. This characteristic can be utilized to develop new methods[22, 23] for surface reconstruction, as contours can be obtained in certain domains, such as point clouds from scanners or height curves of landscapes.

However, contours present some problems that do not always arise with other types of datasets, namely the correspondence problem and the branching problem. The branching problem refers to the challenge of handling contours that merge or separate in subsequent stages of contours. The correspondence problem is the difficulty of matching subsequent contours together. While it is simple when only two consecutive contours are present, it becomes more complicated when multiple contours are present on the same stage. Some researchers choose to avoid these problems by assuming that they will not occur in their domain[24] or by pre-processing to remove problematic areas[25].

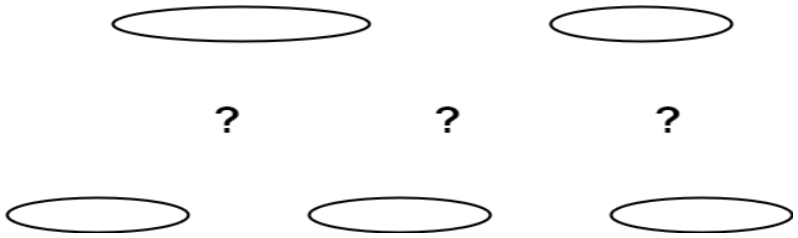


Figure 1.2: Example of branching and correspondence problems between five contours

To take advantage of contours, many methods have been proposed, ranging from simple ones that create a kind of "headband" between two stages of contours[26] to more complicated methods. One example is the use of multi-resolution contours[27], where lower-resolution polygons are used to construct a tiling for the full-resolution polygons. Other researchers use more mathematical tools like differential equations[28], where the solution is efficiently calculated from distance transformations. Eulerian[29] approaches have also been considered, although resolving the problems described before is the most difficult part. Many ideas can be created to reconstruct surfaces with sometimes having the branching problem as center of attention[30–33].

Another idea is to use Delaunay triangulation[34] to reconstruct bands between two stages of contours. The idea is to merge contours from two different heights on the same plane and apply the Delaunay method to it. The correspondence problem is automatically solved, and the authors imagine a more mathematical solution to solve the branching problem. If, during the process, a triangle is formed with two points belonging to different contours with the same height, an intermediate point is created and added to the process to avoid losing details.

1.3 Specific method for surface reconstruction

1.3.1 Poisson surface reconstruction algorithm

Poisson Surface Reconstruction is a widely used and effective method for creating 3D models from point clouds. First proposed by Michael Kazhdan and Matthew Bolitho in 2006[35], it has since become a reference method in the field and is now included in many software and libraries, such as Computational Geometry Algorithms Library (CGAL). The method is based on the resolution of Laplacian equations with appropriate boundary conditions. The main idea is to create a signed distance function that measures the distance between each point of the point cloud and the surface of the object. The method can handle noise and holes in the data, and can produce good results even with a small number of input points. It can also create new points to improve the final result. The output of the algorithm is a smooth and continuous 3D shape of the original point cloud data. However, it should be noted that the method can require significant computational resources, especially for larger datasets. Additionally, implementing the method can be challenging as it requires the normals to the points, which may not always be available in the input data.

1.3.2 Ball pivoting algorithm

The Ball Pivoting Algorithm (BPA) is a surface reconstruction method that uses a ball rolling over a point cloud to reconstruct the surface. It was first presented in 1999 and has the advantage of being simple to understand, although the implementation can be challenging[36]. To summarize how BPA works: Since three points are needed to create a triangle, the method starts with a sphere of a specific radius that only touches the points of a seed triangle. The ball pivots around an edge until it engulfs three points, forming a new triangle. This process is repeated until no triangles can be created from the seed. If all the points have not been processed, another seed is chosen, and the process is repeated.

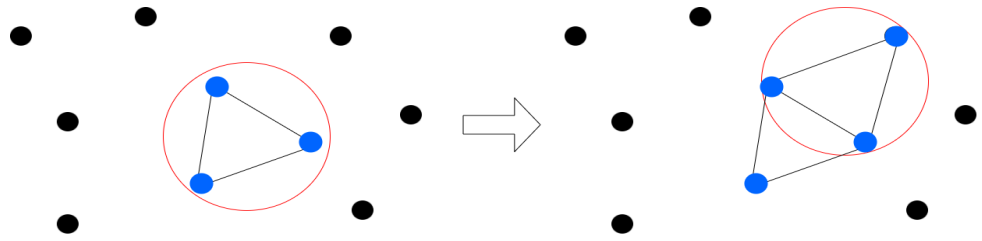


Figure 1.3: 2D representation of the Ball pivoting algorithm

The complexity of the algorithm is linear, according to the authors of the method, and edge cases have been considered. The radius of the sphere can be modified to handle specific parts of a model or uneven densities in the point cloud. Changing the radius allows for the avoidance of holes in the model and loss of details, but choosing the right radius can be challenging because every model is different, although an estimation of an appropriate radius can be made[37].

Since the algorithm is not new, many ideas have been proposed to improve its results[38, 39]. Some ideas include improving pre and post-processing to handle more various point clouds. Multiple instances of BPA could also be a promising vision of the method. One instance for filtering a point cloud and another time to reconstruct a surface on the new dataset[40]. Another idea is to parallelize the process[41]. If the points are grouped into cells, multiple threads can work simultaneously on different cells. The computation time will be slower, but it could be difficult to fuse the results of two threads. To reconstruct a model from contours, another kind of ball like a rugby ball could be used. However, this modification would require significant changes to the mathematical part of the algorithm. Despite the allure of the concept, the comprehensive reconstruction of nearly all aspects would be imperative since the ball would not have a homogeneous shape and would not rotate in the same way.

1.3.3 Marching cubes algorithm

The Marching cube algorithm is also a method which is easy to understand but which is also easy to implement. The method can handle correspondence and branching problems and holes in data can be handled too. This algorithm is not new, as it was first presented at the SIGGRAPH conference in 1987

by William E. Lorensen and Harvey E. Cline[42]. They were working at General Electric, where they needed to efficiently visualize data from CT and MRI devices. The method is similar to the marching squares algorithm, but it is a 3D version of it.

The algorithm operates on a 3D matrix, which is generally created by combining 2D images in an array. These images can be scans, bitmaps, or masks that have been drawn or obtained from other software or devices. They do not need to have the same size, but the process is simplified if they do. The algorithm divides the matrix into smaller cubes and applies a process to potentially draw triangles inside each cube. Each triangle is grouped to recreate the final mesh.

To determine the optimal number of triangles to create for each cube, a lookup table is used. Since there are eight corners in a cube, and the value at each corner can only have two possible results (belongs to an object or not), there are 256 possible situations for a cube. If mirroring and rotations are included, the number of possibilities is reduced to 14, but it is easier to keep a lookup table with 256 possibilities. Each cube is simulated with a model, which is shown on the right below. This model makes it easier to label each edge and corner. With the eight corners, we can obtain the binary value of the index of the situation in the lookup table. On the left below, is displayed the first lines of this lookup table. On each line numbers can be grouped by three, where each number indicates the middle of the edge on which to create a point in that situation. With these three points, a triangle can be created.

```
TRIANGLE_TABLE = {
    {},
    { 0, 8, 3 },
    { 0, 1, 9 },
    { 1, 8, 3, 9, 8, 1 },
    { 1, 2, 10 },
    { 0, 8, 3, 1, 2, 10 },
    { 9, 2, 10, 0, 2, 9 },
    { 2, 8, 3, 2, 10, 8, 10, 9, 8 },
    { 3, 11, 2 },
    { 0, 11, 2, 8, 11, 0 },
    { 1, 9, 0, 2, 3, 11 },
    { 1, 11, 2, 1, 9, 11, 9, 8, 11 },
    { 3, 10, 1, 11, 10, 3 },...}
```

```
//             v7_______e6_____________v6
//              /|                    /|
//             / |                   / |
//          e7/  |                e5/  |
//           /___|______e4_________/   |
//        v4|    |                 |v5 |e10
//          |    |                 |   |
//          |    |e11              |   |
//        e8|    |                 |e9 |
//          |    |_________________|___|
//          |   / v3      e2       |   /v2
//          |  /                   |  /
//          | /e3                  | /e1
//          |/_____________________|/
//           v0         e0         v1
```

The way that the algorithm will simulate the cubes to obtain a closed surface can be difficult to visualize. Assuming that the input data is an array containing 2D images. Each image represents a slice of the object which has to be reconstructed, and is a bitmap in which a pixel with a value of 1 indicates that it belongs to the object, while a pixel with a value of 0 indicates that it doesn't. The algorithm begins with the first two images in the array. It selects four pixels/points on the first slice and their corresponding four pixels/points on the second slice, as shown in the Figure 1.4:

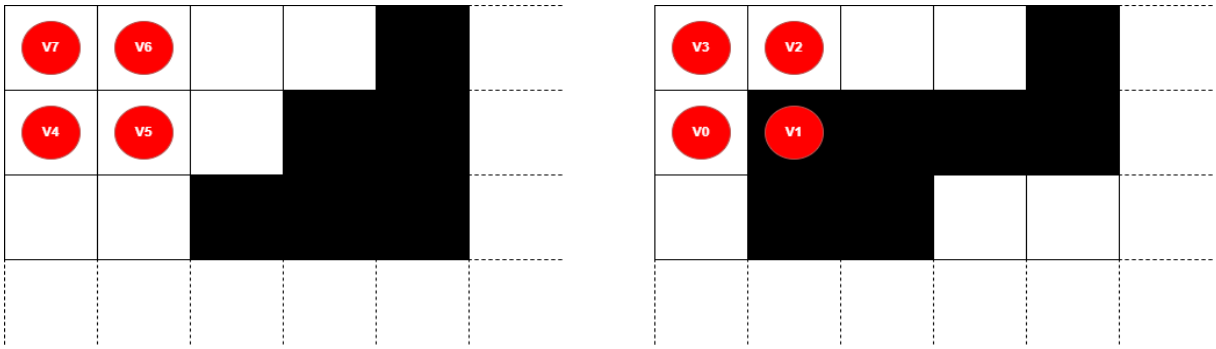


Figure 1.4: Example of simulated cube on two slices of the matrix

Using these eight points, a small cube can be simulated within the larger matrix. Once this element is processed, the method takes the following cube in the row which shares 4 corners and 4 edges with the

previous one. This process is repeated for each consecutive row and each consecutive image in the array to create the final model.

However, the method suffers from some drawbacks. The process can be slow if the size of each image is large. Additionally, the result may be cubic and not smooth. The input data also needs to be sorted but mapping points on the image can be a solution to this problem. Marching cubes is an elegant method , but it may require pre and post-processing depending on the requirements of the problem.

1.3.3.1 Marching cubes algorithm with interpolation

An improved version of the Marching Cubes algorithm that has evolved over time is the incorporation of interpolation to address some of the challenges encountered in the original method. In the current version, the triangles created within each cube are positioned in a predefined manner. This limitation arises from the use of a bitmap, where it is challenging to obtain positional information when elements are represented by only 0 or 1.

To enhance the method, density arrays are employed to replace the bitmap while maintaining a similar objective. Instead of binary values (0 or 1), each element in the density array is assigned a value that aids in determining the position of a triangle. In the Figure 1.5 , a 2D representation of the marching cubes, known as marching squares, is showcased to demonstrate how interpolation can alter the final outcome. On the left, is displayed the original version, while on the right, the result for a marching square implementation with interpolation can be observed. Additionally, since the density array can accommodate a range of values, threshold functions prove useful in selecting only the values relevant to the model.

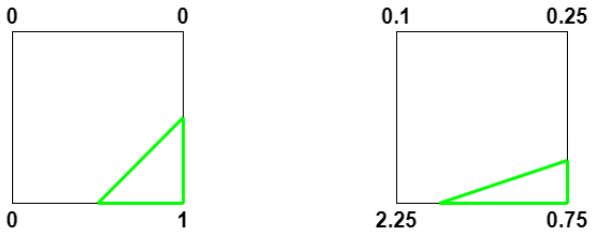


Figure 1.5: Example of Results for marching square algorithm

1.4 Files format

Having explored the intricacies of the algorithmic part in this chapter, it is essential to now shift the attention towards the description of file formats commonly encountered within the medical field. A comprehensive understanding of input and output files is necessary as this serves as the basis for choosing and using the different methods.

1.4.1 Output formats

.obj files

The .obj file format was originally created by Wavefront Technologies for use with their Advanced Visualizer animation software. Over time, the format has become widely adopted by many software applications and vendors in the 3D graphics industry due to its versatility and flexibility. The format is a text-based format that stores information about the position of each vertex of a 3D model, as well as the UV position of each texture coordinate vertex, vertex normal and the faces that make up each polygon, defined as a list of vertices, and texture vertices. This makes it possible to export complex 3D models with multiple materials, textures, and components as a single file. However, it's important to note that the .obj format does not contain information about units, which can create challenges when importing or exporting models between different software applications. Additionally, while the format is useful for static models, it does not support animation or rigging data, which may require the use of additional file formats.

The file contains a list of elements. As we can see in the Table 1.1, the points, texture coordinates, and normals are defined independently. Then, faces are created by giving the indices of the elements we have defined previously. The Table 1.1 helps to understand how a .obj file is created:

Name of the element	Example
Commentary	#Example
Name of the object	o object
Vertex	v 1.0 0.0 -1.0
texture coordinate	vt 1.0 0.25
normals	vn 0.0 -5.0 1.0
face	f 1/1/1 2/5/2 4/6/3

Table 1.1: Description of elements in a .obj file

.stl files

The STL file system is widely used to represent 3D models of various origins. It is easy to create a STL file from data and due to its popularity, many 3D viewers or software for 3D printers accept to treat STL files. That kind of file was developed by 3D Systems and originally released in 1987. At the beginning, the STL files didn't accepted the negative coordinates, but this restriction is no longer enforced in actual standards today. Even if the format had an evolution, the files only represent a 3D model without any other information. It is not possible to know about textures, value of scale units or other common CAD model attributes. Also, the 3D models represented in the files don't need to be closed. Each facet doesn't know which of its vertices is shared with another facet.

The 3D model that is described in the file is composed of triangles. For each triangle, the Cartesian coordinates (x, y, z system) of the three vertices are ordered in a counter clockwise order. Additionally the value of the normal vector are also given. It's also possible to give some other information to STL files like color but that leads to non-standard variations of the files that could be treated abnormally by usual software. The STL files can be written in 2 different representations. The ASCII STL and the Binary STL.

In the ASCII STL. Each file begins with the line *solid name_of_the_model* and finishes with the line *endsolid name_of_the_model*. Between those lines the same pattern for each triangle will be present where each v or n is a floating-point number.

facet normal n_i n_j n_k	facet normal -0.411 -0.552 -0.725
outer loop	outer loop
vertex $v1_x$ $v1_y$ $v1_z$	vertex 125.1 239.1 338.4
vertex $v2_x$ $v2_y$ $v2_z$	vertex 124.6 239.3 338.5
vertex $v3_x$ $v3_y$ $v3_z$	vertex 124.8 239.7 338.2
endloop	endloop
endfacet	enfacet

Table 1.2: Description and example for a triangle in a ASCII STL file

Type of data	**Name of the section**	**Number of bytes**	**Example in hexadecimal**
UINT8[80]	Header	80 bytes	6e6f 7468 696e 6700 0000 0000 0000 0000 0000 0000 0000 ... 0000 0000 0000 0000 0000 0000
UINT32	Number of triangles	4 bytes	78fb 0200
	for each triangle	50 bytes	
REAL32[3]	Normal vector	12 bytes	3e94 a7be 7727 81bf 654d 16be
REAL32[3]	vertex 1	12 bytes	4b07 ff42 2fce bd42 f6e8 6843
REAL32[3]	vertex 2	12 bytes	e1ae fd42 e73d be42 f6e8 6843
REAL32[3]	vertex 3	12 bytes	4b07 ff42 e73d be42 f668 6743
UINT16	Attribute byte count	2 bytes	0000

Table 1.3: Description and example for a binary STL file

The binary version of the STL file allows reducing the memory used for the model without changing the shape. A complex one could rapidly lead to a file of tens of Megabytes. Depending on the number of decimal places, a reduction by a factor 3 to 6 for the size of the binary version compared with the ASCII one is possible. Like the ASCII version, patterns are present in the file like it's displayed in the Table 1.3. The "Attribute byte count" is not useful and is usually set to 0.

1.4.2 Input Formats

DICOM

Digital Imaging and Communications in Medicine (DICOM) is a file format used to manage and exchange medical informatic data, especially medical images. The standard was first created in 1985 by the American College of Radiology and the National Electric Manufacturers Association to facilitate data exchange between different imaging machines from various manufacturers. DICOM is an open and free file format, which makes it accessible to all healthcare professionals, regardless of the equipment and software they use. There are also many free DICOM software libraries and tools that facilitate the implementation of the standard.

Since its creation, DICOM has evolved into a widely adopted standard used worldwide to store, share, and transmit digital medical images, including x-rays, MRIs, CT scans, and ultrasounds. Over time, DICOM has been extended to include new features such as volumetric data storage, perfusion images, nuclear images, videos, and patient monitoring data. The standard has also been adapted for use in fields such as teleradiology, interventional radiology, and radiotherapy. Recently, DICOM has also been extended to encapsulate files like .obj or .stl and their related data.

Each data element in a DICOM file is stored with a tag and a value. The tag is composed of two hexadecimal numbers where the first one describes the group of the value and the second one describes the element. Since DICOM is a standard, it's important to define each tag correctly to avoid mistakes or errors that can be caused by humans or computers. Hundreds of tags can be present in a single DICOM file, some of them are mandatory for each variant of a DICOM file. For example, The Table 1.4 displays some tag definitions and the value that could be assigned to it. VM is for Value Multiplicity, or the number of value allowed for this tag. And VR is for Value representation:

Tag	**Name**	**VM**	**Type**	**VR**	**Example of value**
(0010,0010)	Patient's Name	1	2	PN	Thibault Gregoir
(0010,0022)	Type of Patient ID	1	3	CS	TEXT
(7FE0,0010)	Pixel Data	1	1C	OB or OW	...
(0008,0060)	Modality	1	1	CS	M3D

Table 1.4: Examples of Tag in a DICOM file

Aside from the value and tag, there are other elements that help to understand what kind of data is stored in a tag and its length. It's important to note that each tag has its own defined data type, such as integers, strings, or even a sequence of other tags. Therefore, it's essential to correctly assign the appropriate data type to each tag to ensure proper data interpretation. Overall, the correct use and interpretation of DICOM tags are critical in medical imaging and informatics, where accuracy and precision are vital in patient care.

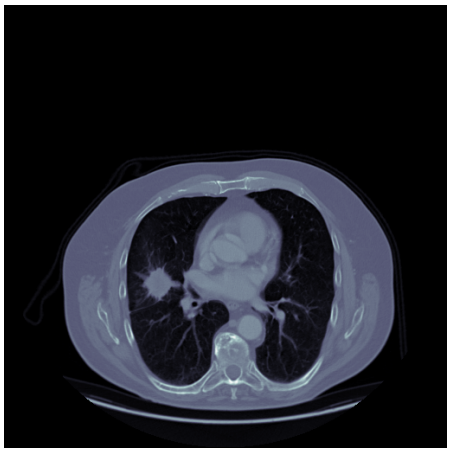
The benefits of DICOM include its ability to integrate data from multiple sources, including images, patient information, and medical records. This integration enhances the accuracy and efficiency of medical diagnosis and treatment. The standardization of DICOM also enables healthcare professionals to exchange data seamlessly between different imaging machines, software, and locations, making it easier to provide the best possible care to patients. Moreover, the extension of DICOM to encapsulate other file formats enhances its flexibility and potential applications.

DICOM CT Scan

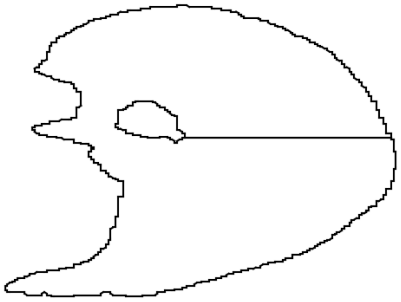
Computed Tomography (CT) scan is a medical imaging exam that uses X-rays to produce detailed images of the internal structures of the body. CT scan images are particularly useful for detecting bone fractures, tumors, and diseases of internal organs, without the need for invasive procedures. This method is quick and non-invasive, making it a preferred imaging option for many medical professionals.

Moreover, CT scans are less sensitive to patient movements than other imaging systems, such as magnetic resonance imaging (MRI). This means that the results are more often clear and accurate, even if the patient moves during the scan.

The images produced by CT scans can be stored in DICOM files, which allow for the storage of additional information, such as imaging parameters, patient data, and metadata. This feature contributes to the ease of sharing the images between healthcare professionals and hospitals, enabling better collaboration and treatment options for patients. The Figure 1.6a is an example of a CT Scan present in a DICOM File.



(a) Example of a CT Scan



(b) Example of a drawn contour from a DICOM RT Struct

Figure 1.6: Example of content present in DICOM files

DICOM RT Struct

The DICOM Radiotherapy Structured Set (RT Struct) is a standard format for transferring patient structure and associated data sets in radiation therapy. It was developed to enable the transfer of contours drawn by medical technicians or artificial intelligence (AI) on CT scans, which are used to highlight organs or specific areas of interest in the images. The contours are stored in a list of triplets, which allows users to retrieve the Cartesian coordinates of each point in the contours. Additionally, each contour has associated information such as the number of points and the geometric type of the contour, which can be a single point, open planar, or other shapes.

The DICOM RT Struct also includes a reference to the ID of the CT scan on which the contours have been drawn, which enables the integration of the contours with the corresponding CT images. This feature is particularly useful in radiation therapy planning, where precise targeting of tumors and organs at risk is essential for successful treatment.

Overall, the DICOM RT Struct is an important tool for facilitating the transfer of radiation therapy-related data, including contour information, between different imaging and treatment planning systems. It enables more efficient and accurate treatment planning, which ultimately benefits patients undergoing radiation therapy.

DICOM STL Encapsulated

The DICOM STL encapsulated file is a variant of the DICOM format used to store 3D models of anatomical structures such as bones, organs, and tumors. The main purpose of this variant is to provide

a context to the models and to take advantage of the importance of the DICOM format in the medical environment.

One of the advantages of this variant is that it allows for easy integration of 3D models with other medical imaging data, such as CT or MRI scans. It is also possible to store additional attributes related to the model, such as the name of the file, the length, and other metadata. However, the ability to manage 3D models is not widely used and few software programs support it.

However, attention must be paid to a fact. The STL encapsulated file does not add a lot of information beyond the 3D model itself. For example, if a series of CT scans includes DICOM STL encapsulated files for every bone, without additional information beyond the model, distinguishing between two vertebrae could become difficult. The encapsulated format proves valuable in offering a standardized framework for 3D models in the medical imaging domain. Saving the data in the DICOM STL encapsulated format is not overly complicated as it simply requires saving the STL file as a binary stream into a specific tag. Note that a similar variant of the DICOM format exists to encapsulate .obj files.

Tag	Name	VM	Type	VR	Example of value
(0042,0011)	Encapsulated Document	1	OB	1	...
(0042,0012)	MIME Type of Encapsulated Document	1	LO	1	model/stl

Table 1.5: Most important tags for a DICOM STL Encapsulated

NIfTI

The Neuroimaging Informatics Technology Initiative is a file format used to save and manipulate medical images. Evolution of Analyze 7.5 file format, the NIfTI format was introduced in 2004 to solve some limitations of the previous standards. The main improvement is the addition of fields.

The NIfTI file format can store multiple 3D and 4D images in a single file, which makes it efficient for storing time series data. It can also store header information describing imaging characteristics, such as spatial resolution, spatial orientation, intensity units, and patient information.

This kind of files will not contain as much information as a DICOM file but will allow to save many images in one file while generally, a DICOM CT Scan will only have one image per file. The two formats are widely used in the medical field but the use of one kind and not the other depends on the context of the task.

1.5 Techniques of Segmentation

Except for photogrammetry, all the methods that have been presented use input data which are already segmented or obtained by various methods. It's important to study those methods to improve the surface reconstruction of models and to understand which techniques can improve a project. At this point, it's important to distinguish two main classes of techniques. The first one uses algorithms to perform segmentation and the second one uses Artificial Intelligence (AI) (like CNN, 3D U-Net...) to create contours or masks from scans. This difference is important because algorithms techniques are presents from years and do not always perform well while using AI is quite new. And with the development of new technologies, results of AI will improve logically with time. So, we have to analyze what was already tried to show what AI could bring.

1.5.1 Algorithm Methods

Thresholding

The techniques using thresholding will create a mask by taking advantage of the contrast in the image of a scan. The idea of the technique is to take values which could be arbitrary (hard thresholding) or calculate with techniques like the Expectation Maximization technique (Soft Thresholding) from the image and say which pixels belong to the object and others not by creating kinds of classes due to the value of a pixel. With this idea outputs can easily be created, but the results will not be perfect due to noise and

irregularity in the image even if smoothing like a convolution can be applied on the scan. To differentiate part of the body the Hounsfield scale can be used. It is a quantitative scale for describing radiodensity. By taking a pixel of a scan with an average linear attenuation coefficient µ. The Hounsfield value can be calculated with the formula of the Equation 1.1:

$$HU = 1000 \times \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{1.1}$$

Where μ_{water} and μ_{air} are the linear attenuation coefficients of water and air. And by taking tables with well known values like some examples are displayed in Table 1.6, different parts of a body can easily be contoured.

Substance	**HU**
Air	-1000
Fat	-120 to -90
Bone Cancellous	+300 to +400
Bone Cortical	+500 to +1900
Water	0

Table 1.6: Hounsfield Unit Chart

Clustering

Those techniques will often operate on human's decisions. The goal of those methods is to determine which class a pixel belongs to. But, to perform that, the methods have to know the number of clusters and the final stability depends on initialization.

K-means is a well-known method in unsupervised learning used for creating clusters. The process starts by initializing K centers for the clusters. This initialization can be done randomly or based on predefined criteria. The algorithm then iteratively performs the following steps: each data point is assigned to the closest center. The assignment can be based on Euclidean distance or other mathematical techniques. Once all the points have been assigned, the cluster centers are updated, typically by calculating the mean of each cluster. The new center becomes the reference for the next iterations. This process continues until convergence or until certain predetermined criteria are met.

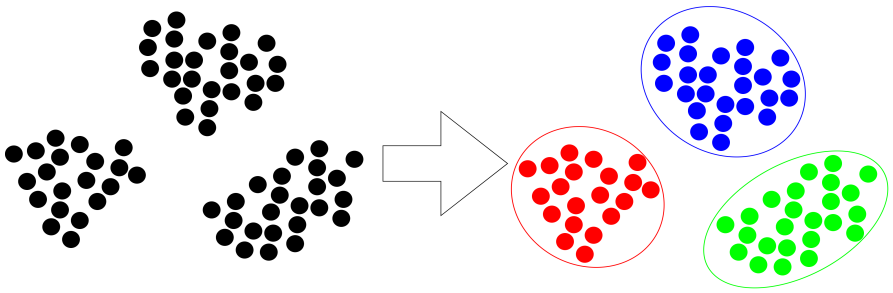


Figure 1.7: Example of Results after K-means algorithm

The mean shift algorithm will resolve some problems of K-means because it does not require prior knowledge of the number of clusters but it still needs humans to class the results. Clustering methods are essentially non supervised. From those, similar methods to classify pixels can be imagined. But labels have to be provided to some pixels and other pixels will be given their label according to this prior knowledge. The more known methods are Parallelepiped classifier, minimum distance to centroids, gaussian methods or K nearest neighbors. But those methods are not perfect and to use them in a project, some kinds of pre/post processing need to be applied to reach a specific goal.

Region-Growing

As the previous ones, Region-Growing methods are quite simple to understand. The algorithms start from a seed and will expand the detected regions by looking if the neighbours' pixels of the detected

regions satisfy some conditions to know if they belong to the object or not. The method stops when no pixel can be included anymore. It has the advantage to be intuitive and to produce connected regions and it could be easily parallelized. The biggest default of this technique is the seed which is "use-dependent". Body will not be exactly the same between two scans of two patients or two scans of the same person. Finding a technique which will generate a seed automatically should not be a priority when an AI could be more efficient.



(a) Schema of Region Growing method from seed

(b) Schema of Active Contours method

Figure 1.8: Example of segmentation done by algorithms

Active contours

When Region-Growing algorithms start with a seed on the images. Active contours methods start with a shape which encompass the region that we want to contour. This initial shape is deformed to match the shape of an object. This deformation is obtained by applying a minimization of an energetical function which takes into account the internal and external constraints. Internal constraint terms keep the curve smooth and continuous, while external constraint terms pull the curve to the edges of the object based on image characteristics, such as brightness, gradient, and textures.

Those methods are iterative and the process repeats until the convergence is achieved to obtain the final shape of the object. They are used in many domains and not just medical imaging but there are some problems. A user has to define the constraints and those could be different depending on the scans, the organs... And only contours can be obtained. It's difficult to have a 3D version of the algorithms.

1.5.2 AI Method

U-net

The methods described in the previous sections may depend on user input and preferences. Software such as 3D-Slicer, Osirix, or ITK-Snap provide algorithms for generating contours or 3D models, but optimizing or generating multiple models quickly can prove challenging. This is where artificial intelligence can be advantageous in tackling the task and particularly U-net.

The U-Net is an elegant machine learning technique originally used for image segmentation in the medical field. It was designed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 and reported in the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation"[43]. However, the idea is not entirely original since it is based on a previous work on fully convolutional networks (FCNs) proposed by Long, Shelhamer, and Darrell in 2014. Nevertheless, the U-Net network achieves better segmentation results using fewer training images than previous methods. Originally designed for a medical area, those kinds of AI are now used in many domains like autonomous vehicles, semantic segmentation, object detection...

The name "U-Net" comes from the architecture of the network. It consists of a contracting path followed by an expansive path. The network applies repeated convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation during the contraction phase. The goal is to reduce spatial information while increasing feature information. In the second phase of the process, the network

combines feature information and reduced images using a sequence of up-convolutions and concatenations to obtain images of the same size as the input but with the desired information, such as masks. The Figure 1.9 is a graphical representation of a U-Net.

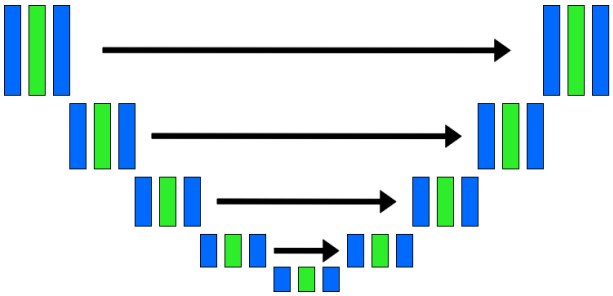


Figure 1.9: Graphical representation of a U-net

nnU-nets

U-Net has become a reference in artificial intelligence techniques, widely used as a base to develop new methods for specific designs. One such method is nnU-Net, which was published in Nature on December 7th, 2020 by five authors[44]. This new technique is a deep learning-based segmentation method designed to automatically configure itself, including pre and postprocessing, training, and network architecture.

To achieve this automatisation, the authors worked on three important elements. First, they collected design decisions that were global across datasets and robust enough to be fixed parameters for many adaptations. Second, they formulated heuristic rules, in the form of explicit dependencies between design choices and dataset properties, for as many remaining decisions as possible. These are known as rule-based parameters. Finally, the remaining decisions were learned only from the data, known as empirical parameters. This procedure allowed for an automated configuration without any manual decisions. The authors optimized the process to be labeled as "fast" and requiring no more resources than other techniques.

The technique involves creating three distinct U-Net configurations using different methods: a 2D U-Net, a 3D U-Net that works with the entire image resolution, and a 3D U-Net cascade where the initial U-Net functions on reduced-resolution images, and the second U-Net, improves the segmentation maps generated by the first at full resolution. Through cross-validation, nnU-Net selects the most effective configuration or combination. Additionally, nnU-Net selects 'non-largest component suppression' as a post-processing step if it results in performance improvements.

nnU-Net is capable of performing 53 segmentation tasks on medical images, and it outperforms other methods on 33 structures. For the other 20 structures, the results are close to the leading methods.

TotalSegmentator

TotalSegmentator is an AI developed by researchers from the Department of Research and Analysis at University Hospital Basel[45]. The initial goal of developing this AI was to build a model that could detect a large number of organs. While existing models were available, only the model developed by Chen et al. was the closest to their goal, as it was able to detect 50 different classes. The researchers also wanted to make their model and dataset freely accessible to others. As such, they developed a model that solved three problems:

- The model can segment a wide range of data structures (more than 100)
- The model is trained on a large clinical dataset
- The model and training data are publicly available and easy to use.

To create their AI, the researchers used the nnU-Net described before[44]. However, they made two modifications. First, they removed the "mirroring" parameters for data augmentation as enabling these parameters made it difficult to distinguish between symmetrical bones of the body. Second, they increased the number of epochs since the size of the dataset was larger.

The main task of TotalSegmentator is to segment 104 classes. However, doing this all at once requires a lot of memory and resources. To address this, the researchers split the classes into five parts of approximately 21 elements each and trained five nnU-nets. To train their model, they used a process called "active learning." They manually segmented the first five subjects, then trained the nnU-Net, and corrected only its predictions. They repeated this process with 20, 100, and 1000 subjects, where the initial segmentations were done by the AI. This method allowed them to use fewer humans for segmentation and give better results for each training.

To evaluate their model, the researchers used the Dice score and normalized surface distance (NSD). They also compared their results to those from the "MultiAtlas Labeling Beyond the Cranial Vault Challenge." However, this comparison only provided labels for 13 elements. The results displayed in their article allow us to see that their method performed better than previous leading models.

Chapter 2

Problem statement and objectives of the master thesis

The main objective of this Master's thesis is to explore the clinical interoperability standards that can be used for 3D printing of medical organs. To achieve this goal and to validate the proposed solutions, 3D models will be generated from medical data to provide real-world examples.

This objective can be divided into multiple subtasks, with the primary task being to develop the ability to create 3D models from DICOM RT-Struct files and to understand the techniques required for this task. As outlined in the State of the Art chapter, various algorithms exist to reconstruct models. However, the challenge lies in identifying the right techniques to create complete and realistic models and addressing any issues that may arise during the process.

Once the first task is completed, it will enable exploration of other objectives, including the ability to print the 3D models generated previously, ensuring that the models are suitable for 3D printers, and determining whether the render will match the prints. Additionally, identifying the global constraints involved in this task will be crucial.

In parallel, another sub-task involves the ability to recreate DICOM files that contain the 3D models and developing a web application capable of displaying these models in a virtual environment. The results of the previous task will be helpful in this sub-task since the creation of DICOM STL Encapsulated requires the combination of input data and the result of model reconstruction.

A third task will involve analyzing the potential of artificial intelligence (AI) to perform segmentation on CT scans. If the results are conclusive, the use of AI to create 3D models from CT scans without creating DICOM RT-Struct files will be explored to determine whether the algorithms created for RT Struct files can be reused for this task or whether other solutions need to be developed. An implicit goal of this task is to create a pipeline that can generate 3D reconstructions of the human body with minimal human interaction.

Finally, all of these steps will be integrated into a user-friendly and open-source software, enabling anyone to utilize the codes developed for this master's thesis. The software will be designed to be intuitive and accessible, ensuring that even individuals with limited computer usage experience can harness the potential of 3D reconstruction for personal and community purposes.

Chapter 3

Methodology

The purpose of the "State of the Art" section was to provide an overview of the various knowledge and techniques employed in 3D model reconstruction, with a particular focus on applications in the medical field. This knowledge serves as a foundation for achieving a specific goal. In the upcoming "Methodology" section, the techniques and methodologies employed for model reconstruction will be elucidated. Furthermore, additional technologies utilized to accomplish specific objectives will also be discussed.

3.1 First look at the data

In the medical imaging field, it's common to use DICOM RT Struct files to store contours derived from medical images. The data can be extracted by looking at the appropriate tags.

The DICOM format has standard guidelines for storing contours of 3D models, which can be found in the tag *(3006, 0039) : ROI Contour Sequence*. Within each item of the sequence, two tags can be found : *(3006, 002a) : ROI Display Color* and *(3006, 0040) : Contour Sequence*, which is also a sequence of items. In this new sequence, we can find the tag *(3006, 0050) : Contour Data*, which contains an array of floats with the Cartesian coordinates of each point. As described in the State of the Art section, the array is displayed as follows: $[X_1, Y_1, Z_1, X_2, Y_2, Z_2, ...]$.

It's worth noting that DICOM RT Struct files can contain multiple 3D models, each with its own contour sequence. To extract all the data for the models stored in an RT Struct file, we just need to iterate through the ROI Contour Sequence and extract the Contour Data for each item.

After extracting the contour data from the DICOM RT Struct files, we can analyze the arrays of points. It's important to note that all points in a contour have the same Z value, but two contours from the same model can be located at the same height. The way contours are stored may vary depending on the file's creator, but this should not complicate the process too much. However, awareness of this variability is critical for processing.

Another important consideration is that contours are only linked to the original scans, so we won't have information about what's between two slices (typically 5 mm apart). This can result in situations where points are closer to their neighbors off the same height than to their "corresponding" points on contours from the next slice. This can affect the accuracy of the reconstruction and should be taken into account when selecting an algorithm.

It's also worth noting that the contour data only contains information about the location of the points, and not their orientation or any other attributes, such as the normal vectors. Moreover, there is not necessarily a predefined order for the points in an array, so we have to ensure that the algorithm used for reconstruction is robust to these factors. Finally, the quality of the original medical images and the accuracy of the segmentation process can also impact the accuracy of the resulting 3D model. Input data could lead to some post-processing to upgrade the models but this detail is difficult to analyze generally and can just be done case by case.

3.2 Choice of the algorithm

The most important choice of this work is which algorithm will be used. In the State of the art section we learnt a lot of methods and points of interest to realize this project. Some parts were more explained than others due to their potential. To resume the actual situation is that with DICOM RT Struct files, we obtain a specific kind of cloud point data. Indeed, the points are organized by contours and are only present at some Z values. Photogrammetry and other methods which use data from images to reconstruct models can be ousted. The focus will be on some relevant methods which take cloud points or contours as input.

Poisson Surface Reconstruction

The problem with using the Poisson surface algorithm for 3D models is the absence of normals for the points. As described previously, the original file only contains the coordinates of the points. Therefore, estimating normals for these points requires additional effort, making it more difficult to create than simpler methods.

Pivoting Sphere Algorithm

Given that the data is linked to scanning, there are gaps of several millimeters between consecutive contours. Therefore, it would require considerable effort to modify the existing method by replacing the sphere with a kind of rugby ball, as proposed in the state of the art section. Because, if a sphere is used, a lot of details could be loose.

Create a specific method

To solve the challenge, a custom method can be created. One section in the state-of-the-art chapter discusses the creation of surfaces from contours. Advice and techniques from this section can be used to build a method. However, the difficult part of implementing this method is that it is not always guaranteed that the contours will be sorted in the same order. Additionally, sometimes some contours are concentric but are saved as a single contour, while other times, contours are saved in different elements. The correspondence and branching problems could also be challenging to solve. It is crucial to ensure that a contour is divided into several elements accurately. Although this method could be highly efficient for spherical shapes, it could become slower when the algorithm is generalized. In Figure 3.1 are some examples of results that could be obtained with an algorithm which create surface by creating "headbands" from contours but which don't handle branching and holes in contours. Although these results are promising, the amount of work needed to complete the challenge is significant.

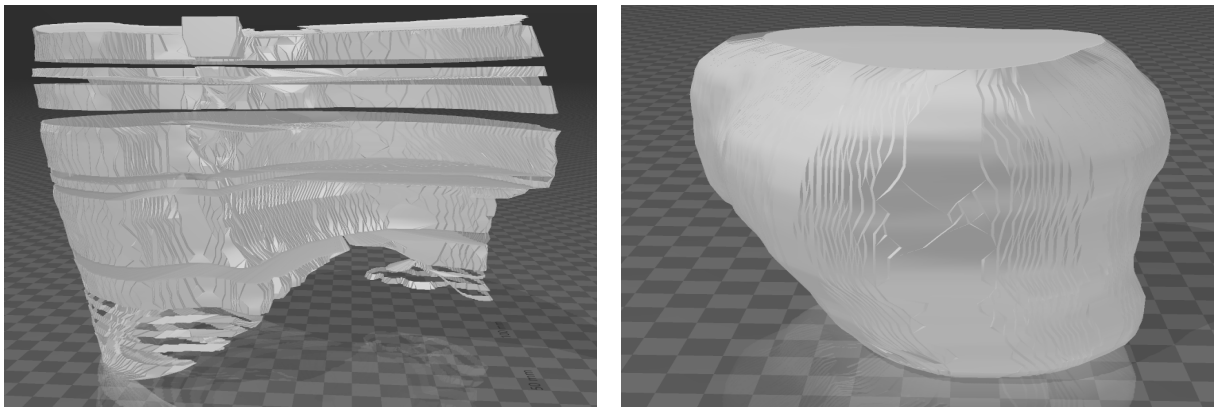


Figure 3.1: Examples of results from an incomplete algorithm

Marching Cubes Algorithm

There are various libraries and methods available in programming languages that enable the drawing of contours on an image and filling the created polygons. These resources, when taken as input, make a perfect fit for the marching cubes algorithm. Furthermore, the shortcomings of this method are widely known, making it easier to generate ideas for optimizing the results. As the method produces triangles

in each cube, a .stl file can be easily created, and due to its simplicity, the method can be translated into many different languages if needed. All of these elements serve to demonstrate that marching cubes is an excellent choice for surface reconstruction.

3.3 Implementation of the Marching Cubes Algorithm

Now that the algorithm has been chosen, we can examine the various constraints for fitting the data for the Marching Cubes. We will need to create a matrix with the different images, but an additional array is needed to keep track of the Z-axis values of the points. This array will have the same size as the image array, with the height of the first image at position 0 in the additional array.

Since the contours will never have the same size and shape between different models, a way to generalize the process must be used to avoid future problems. Therefore, additional information must be kept for the Marching Cubes algorithm itself. The second element that needs to be created is a structure that will store the maximum and minimum values along the X and Y axes for all the points of a model. The goal is to avoid the creation of an infinite bitmap and reduce the computational time of the entire code. Once these values have been filled, a center can be positioned using the formulas 3.1-3.2:

$$Center_X = (X_{max} + X_{min})/2 \tag{3.1}$$

$$Center_Y = (Y_{max} + Y_{min})/2 \tag{3.2}$$

Once the essential elements are obtained, bitmaps can be created. At first, bitmaps are just completely empty arrays/matrices. The method described below allows for variable sizes of the bitmaps. Although some modifications may be necessary to adjust the size based on the values used to calculate $Center_X$ and $Center_Y$. The ability to have a variable size for the bitmap is useful to obtain different types of results. To create the matrix, a bijection will be used between the bitmaps and the different contours. The structure created earlier becomes useful here. By following the formulas 3.3-3.4, each point of the contours can be mapped onto the bitmap. Of course, some details may be lost, but due to the possibility of changing the size of the bitmap, a trade-off can be achieved between size and details.

$$New_X = size \times ((X - Center_X)/(X_{max} - X_{min})) \tag{3.3}$$

$$New_Y = size \times ((Y - Center_Y)/(Y_{max} - Y_{min})) \tag{3.4}$$

Methods are used to draw the polygon contoured by the points and fill it to obtain the final bitmap. However, there is still a problem. Depending on the data and the model being reconstructed, a top and bottom plane may not necessarily be present, resulting in holes in the final model. To address this issue, two empty bitmaps need to be added at the border of the matrix, and the corresponding Z value should also be added to the additional array created before. Those values can easily be calculated using the other Z values in the array. Another point of attention related to this problem consists in the points located at the extreme values on the X and Y axis. They could also lead to holes in the current situation. To solve this, when the method is asked to create square bitmaps of size S, the code will create square bitmaps of size S+2 but will map the points on a square of size S. This is to ensure that an empty border is present to each bitmap, guaranteeing that the final model will be closed.

Because the matrix is completed, the marching cube algorithm can operate on each small cube of size 1 by using loops similar to the pseudocode in Listing 3.1. The cube is simulated by taking the index of each corner. Following this pseudo code, the value of the height corners will be $(L, i, j), (L, i+1, j), (L, i+1, j+1)$... :

```
for L in range(len(bitmap)-1):
    for i in range(len(bitmap[0])-1):
        for j in range(len(bitmap[0][0])-1):
            ...
```

Listing 3.1: loops used for Marching cubes

For each cube, we can determine the number of triangles that have to be created by using the corners as values and looking up the corresponding situation in a lookup table. To create the actual triangles, we reuse the bijection between the bitmap and the real situation. The formulas 3.3-3.4 are used in the opposite way, as shown in the formulas 3.5-3.6:

$$New_X = (X/size) \times (X_{max} - X_{min}) + Center_X \tag{3.5}$$

$$New_Y = (Y/size) \times (Y_{max} - Y_{min}) + Center_Y \tag{3.6}$$

If the values of X and Y represent indexes of the bitmap, it is necessary to subtract half of the bitmap's width from them to obtain the correct mapping. This adjustment is required because the element located at position (0,0) in the bitmap corresponds to the real-world position (-S/2,-S/2), where S represents the width of the bitmap.

All the triangles can be saved in a structure and written in the final STL file once all the cubes have been visited. To create correct STL files, normal vectors for the triangles are needed and the formulas are shown in equation 3.7-3.9. As described, the method is not complicated and generates results without a lot of additional information, while handling many problems that could be present when using other algorithms.

$$N_x = (Y_2 - Y_1) \times (Z_3 - Z_1) - (Z_2 - Z_1) \times (Y_3 - Y_1) \tag{3.7}$$

$$N_y = (Z_2 - Z_1) \times (X_3 - X_1) - (X_2 - X_1) \times (Z_3 - Z_1) \tag{3.8}$$

$$N_z = (X_2 - X_1) \times (Y_3 - Y_1) - (Y_2 - Y_1) \times (X_3 - X_1) \tag{3.9}$$

3.4 Improve the results of Marching Cube algorithm

Two ideas will be explored to improve the results of the Marching Cubes algorithm. These ideas can be combined as they address different parts of the algorithm. The first idea involves investigating the use of interpolation, as explained in the state-of-the-art section, to enhance the algorithm results. The second idea proposes working directly on the reconstructed mesh instead of the original data.

3.4.1 Marching cubes with interpolation

3.4.1.1 Preprocessing

In order to utilize interpolation, a preprocessing step is required to convert the bitmap into a density bitmap. Several approaches can be considered. One global approach is to combine the values of neighboring pixels or elements in the matrix to obtain a mean floating value. This method can be applied in both 2D and 3D contexts. Thus, each non-border pixel becomes the mean value of its neighborhood pixels. The Figure 3.2 demonstrates how pixels can be grouped to obtain a single value. This idea can be extended to 3D by considering the corresponding pixels in the subsequent bitmaps.

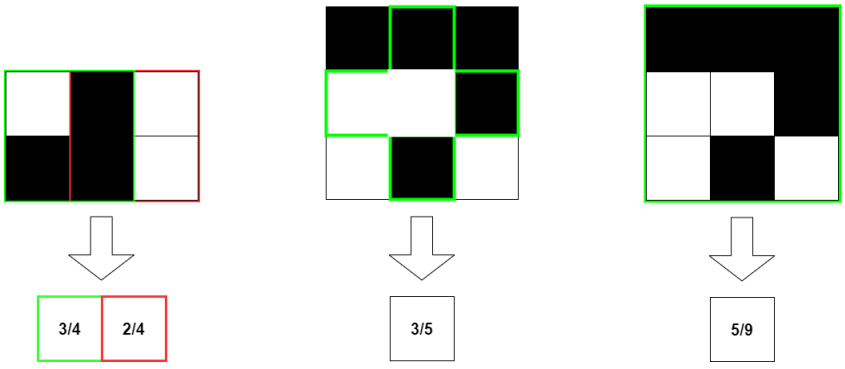


Figure 3.2: 2D examples for way to create density maps

Another way to obtain density bitmap is by using more complex mathematical tools like convolution. Convolution is an operation widely used in image processing and computer vision. The goal is to combine each pixel with his neighbors by using a filter or a kernel matrix. Each element of the kernel is multiplied by the value of the corresponding pixel in the image, then the results are summed to obtain the new value of the pixel in the output image. This process is repeated for each pixel in the image. The goal of the convolution could be border detection, obtaining a vague effect on the image, noise cancellation... Depending of the goal, different kernels exit.

In this work a Gaussian filter will be used to obtain a blurred effect on the bitmap. The idea is to obtain a smoother transition between values and not only 0 or 1. The formula to compute each element of the Gaussian kernel is given by equation 3.10. Even if ,afterwards, a function from Scipy library will be used, an understanding of the different tools is mandatory to understand what will be done by the computer.

$$G(i,j) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{3.10}$$

G(i,j) is the value of the filter at position (i,j) of the kernel. σ is the standard deviation of the Gaussian filter and the fraction in front of the exponential is present to normalize the filter. Many other ideas exist to create a density map. However, it can also be challenging to extract useful information from 2D bitmaps.

3.4.1.2 Modification on the algorithm

Once the bitmap has been obtained, the algorithm can be modified. The overall process will remain essentially the same, but additional elements will be incorporated. First, a threshold function needs to be created to determine which values belong to an object and which do not. Next, the value of each corner needs to be passed to the function responsible for translating points from the density map to their corresponding positions in the real world, similar to the original marching cubes algorithm. However, before proceeding, we need to establish a method for accurately position the points on the edges of the cube by extracting information from the density map. A ratio function can be defined as in Listing 3.2:

```
def ratio(A,B):
    if(A==0.0):
        return 1-B/1.0
    elif(B==0.0):
        return A/1.0
    else:
        return (A)/(A+B)
```

Listing 3.2: Ration function used for interpolation

The way that the points are created is the same as on the original version but the points are not put on the middle of edges but set on the value of a corner + the value of the ratio. This method works because the value of ratio will always be between 0 and 1 since the value in the density bitmap will always be between 0 and 1 too. Those floating values will assure that the point is set on an edge of the cube and not beyond. The main risk with this method is that the points don't move a lot compared to the original version of the marching cube due to the small size of each cube.

3.4.2 Post-processing

Another way to optimize results is to work on the 3D mesh created by the Marching cube algorithm. By moving the nodes and triangles, it's possible to rearrange the model to obtain a smoother one. Some libraries already have implemented such methods. The Visualization ToolKit (VTK) is a software and a library that can be used in python for such problems. Especially the class vtkSmoothPolyDataFilter. Some parameters can be modified as the number of iterations or the relaxation factor which indicates if the point can move a lot or not. The method processes as follows.

The method begins by creating a copy of the input mesh to ensure that any modifications are not applied directly to the original data, but rather to a duplicate. Then, the algorithm enters into an iterative phase. For each point in the mesh, its position is adjusted by performing a weighted interpolation, taking into account the positions of neighboring points and the relaxation factor. The relaxation factor determines the extent to which neighboring points influence the movement of each point. A lower value of the relaxation factor reduces the impact of neighbors, resulting in smoother model. Conversely, a higher value of the relaxation factor gives more importance to neighbors, resulting in more pronounced smoothing effects.

To preserve the overall shape of the model and prevent excessive reduction, points located on the model's borders are constrained to move within a limited range. Once all points have been processed, the new mesh replaces the previous one. This process is repeated for a predefined number of iterations, allowing for gradual refinement of the mesh's smoothness.

This method utilizes Laplacian smoothing, which involves the application of a Laplacian operator to analyze the local geometry variations of the mesh. It is important to acknowledge that Laplacian smoothing may introduce a slight loss of fine detail in the mesh, as it tends to average the positions of the points. Therefore, the parameters used in the method need to be carefully chosen to strike a balance between smoothing the mesh and preserving important features.

To ensure that the resulting models exhibit similarity, a comparison of the volumes can be conducted. If there is a significant difference between the volume of the original and smoothed models, it indicates the need for alternative solutions to address the issue at hand.

3.5 Creation of DICOM STL Encapsulated

Recreating a DICOM file with the data of the STL file encapsulated is not really hard. Due to the standardisation of the DICOM format, the process is really scripted. The result file just needs the right tags with the right values and that's all. The final file can be created from scratch or by taking only the useful value from a DICOM file given as an input. In the Table 3.1, the mandatory tags or tags which are needed for following tools are displayed with the Name, the Value Representation (VR), the Type and an example of value.

Tag	**Name**	**VR**	**Type**	**Example of Value**
(0002, 0000)	File Meta Information Group Length	UL	1	194
(0002, 0001)	File Meta Information Version	OB	1	b'\x00\x01'
(0002, 0002)	Media Storage SOP Class UID	UI	1	Encapsulated STL Storage
(0002, 0003)	Media Storage SOP Instance UID	UI	1	1.2.826.0.1.36800...
(0002, 0010)	Transfer Syntax UID	UI	1	Implicit VR Little Endian
(0002, 0012)	Implementation Class UID	UI	1	1.0.0.0.0.0.22213.1.143
(0002, 0013)	Implementation Version Name	SH	1	0.5
(0008, 0016)	SOP Class UID	UI	1	1.2.840.10008.5.1.4.1.1.4
(0008, 0018)	SOP Instance UID	UI	1	1.3.6.1.4.1.14519.5.2... ...
(0010, 0010)	Patient's Name	PN	2	Thibault Gregoir
(0010, 0020)	Patient ID	LO	2	HF1316
(0010, 0030)	Patient's Birth Date	DA	2	20000101
(0010, 0040)	Patient's Sex	CS	2	F
(0020, 0010)	Study ID	SH	2	753951
(0008, 0020)	Study Date	DA	2	20070119
(0008, 0030)	Study Time	TM	2	101601.934000
(0008, 0050)	Accession Number	SH	2	
(0008, 0090)	Reffering Physician's Name	PN	2	REMOVED
(0020, 000D)	Study Instance UID	UI	1	1.2.826.0.1.3680043.8.4...
(0008, 0023)	Content Date	DA	2	20071004
(0008, 002A)	Acquisition DateTime	DT	2	20230520175351.517563
(0008, 0033)	Content Time	TM	2	164833
(0020, 0013)	Instance Number	IS	1	42

(0028, 0301)	Burned In Annotation	CS	1	NO
(0040, A043)	Concept Name Code Sequence	SQ	2	
(0042, 0010)	Document Title	ST	2	ROI
(0042, 0011)	Encapsulated Document	OB	1	...
(0042, 0012)	MIME Type of Encapsulated Document	LO	1	model/stl
(0008, 0070)	Manufacturer	LO	1[1]	SIEMENS
(0008, 1090)	Manufacturer's Model Name	LO	1	Aera
(0018, 1000)	Devial Serial Number	LO	1	0
(0018, 1020)	Software Versions	LO	1	1
(0008, 0060)	Modality	CS	1	M3D
(0020, 000E)	Series Instance UID	UI	1	1.2.826.0.1.3680043.8.4...
(0020, 0011)	Series Number	IS	1	1
(0020, 0052)	Frame of Reference UID	UI	1	1.3.6.1.4.1.14519.5.2.1...
(0020, 1040)	Position Reference Indicator	LO	2	XY
(0040, 08EA)	Measurement Units Code Sequence	SQ	1	...
>(0008, 0100)	Code Value	SH	1C	mm
>(0008, 0102)	Coding Scheme Designator	SH	1C	UCUM
>(0008, 0103)	Coding Scheme Version	SH	1C	1.5
>(0008, 0104)	Code Meaning	LO	1	STL file encapsulated

Table 3.1: Mandatory tag for a DICOM STL Encapsulated

Once the file is created, tools can be used to verify whether the file is correct or not. The dicom3tools library provides various tools for this purpose. One such tool is "dciodvfy," which can be executed through the command line. This program examines the tags within the file, and if it detects anything suspicious, it generates a warning. It also reports an error if a mandatory tag is missing or if a tag is not properly defined. If no problem is found in the file, the program displays the given DICOM file type as input. In this part of the project, the program should show that the file is a DICOM STL Encapsulated.

3.6 Creation of a webViewer

3.6.1 Schema of the process and presentation of the tools

The DICOM STL Encapsulated format poses a challenge when it comes to finding freely available and user-friendly software for displaying STL files within a 3D environment. While some medical software already include 3D file viewers or extensions, they can often be difficult to use. The objective of this part of the project is to develop a straightforward web application capable of displaying and manipulating STL files encapsulated in DICOM files. To accomplish this, several technologies will prove useful, namely Flask, Axios, and Three.js. Various tools will be utilized to complete the different steps outlined in Figure 3.3. The schema serves as a visual representation of how STL files are presented within a scene, with each element warranting careful consideration during the code creation process.

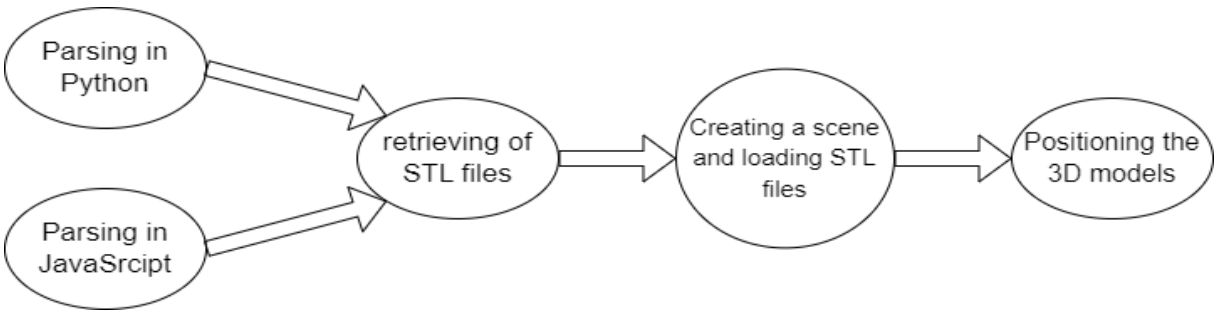


Figure 3.3: Schema of the process

[1]Type 1 in Enhanced General Equipment

Three.js will play a crucial role in creating a scene where 3D models can be rendered. Two components within Three.js, namely OrbitControls and STLLoader, are particularly relevant to this project. The names of these components provide a clear indication of their respective purposes. They facilitate the loading and navigation of 3D models, leveraging pre-developed and optimized technologies to streamline this aspect of the project.

Flask is a micro web framework for building web applications in Python. It is a lightweight framework that provides basic features for web development such as routing, templating, and accessing databases. Axios, on the other hand, is a JavaScript library used for making HTTP requests from a web browser. It is commonly used in front-end web development to interact with web APIs and fetch data from remote servers. Axios provides a simple and consistent API for making HTTP requests and supports features such as request and response interceptors, automatic JSON parsing, ... Flask can be used to create a server-side application that serves the HTML, CSS, and JavaScript files to the client's web browser. Axios can then be used to make HTTP requests from the client-side JavaScript code to the Flask server, which can respond with JSON data or HTML templates.

When uploading files to a Flask server from the web browser, the CORS (Cross-Origin Resource Sharing) policy can sometimes cause problems as it restricts web pages from making requests to a different domain. To overcome this issue, a solution is to simulate a server on the user's computer using the Flask library in Python and the Axios library in JavaScript. This allows the user to upload files directly from their computer to the Flask server without violating the CORS policy.

3.6.2 Parsing the DICOM files and Retrieving of STL files

Parsing in Python

The to_json_dict() function in pydicom proves useful in converting the DICOM STL File to a JSON format with minimal issues. However, a particular difficulty persists. The STL file is encoded as a binary stream within the DICOM File, which introduces the possibility of errors during the conversion to JSON. It is crucial that the binary stream remains intact, as any deviation would prevent the JavaScript code from displaying an accurate model. Additionally, JavaScript utilizes UTF-16 for string encoding, which requires attention when transferring the JSON file. To address these challenges, the implementation of the base64 library in Python offers a comprehensive solution. After employing the following code snippet, the STL and so the DICOM file are saved into a variable reachable by an axios request. The JavaScript code 3.4 can after get the data with an axios. And once the data are obtained, JavaScript can process the byte stream to get the original STL file and put it into a Uint8Array named "bytearray" :

```
    DC_J['00420011']['InlineBinary'] =
        base64.b64encode(DC['00420011'].value).decode('utf-8')
```

Listing 3.3: Python code to encode a byte stream into a base64 array

```
axios.get('/data').then(function(response){
    const answer = response.data;
    let encoder = new TextEncoder();
    let process_data = [];
    for (let i in answer){
          var binarystring = atob(answer[i]['00420011'].InlineBinary);
          var bytearray = new Uint8Array(binarystring.length );
          for (var j = 0; j < binarystring.length; j += 1) {
             bytearray[j] = binarystring.charCodeAt(j);
          }
        process_data.push(bytearray);
    }
});
```

Listing 3.4: JavaScript code to process the base64 array encoded into a JSON

Parsing in JavaScript

This parsing can also be done entirely in JavaScript using different tools. The dicomParser library can be utilized to simplify the process. The code in the Listing 3.5 achieves the same goal as the previous one, but the files are directly handled by the JavaScript code.

```
const reader = new FileReader();
reader.onload = () => {
    const byteArray = new Uint8Array(reader.result);
    const dataSet = dicomParser.parseDicom(byteArray);

    const obElement = dataSet.elements['x00420011'];//Tag of the ByteStream
    if (obElement) {
        const byteSlice = byteArray.slice(obElement.dataOffset,
            obElement.dataOffset + obElement.length);
        process_data.push(byteSlice)
    }
};
```

Listing 3.5: JavaScript code to parse a DICOM file and retrieve an encapsulated STL file

3.6.3 Creating a scene and loading STL files

Once we obtain the STL files, a scene needs to be created to incorporate the models. A code similar to the Listing 3.6 example can be utilized. It provides a clear understanding of where the scene is created, how the models are loaded and utilized to create a "mesh" variable for Three.js, and how to add a mesh to a scene:

```
const scene = new THREE.Scene();
const loader = new STLLoader();
const geometry = loader.parse(byteSlice.buffer);
const material = new THREE.MeshStandardMaterial({
    color: 0xff0000,
    metalness: 0.35,
    roughness: 0.5
});
scene.add(THREE.Mesh(geometry, material));
```

Listing 3.6: JavaScript code to create a scene using Three.js

3.6.4 Positioning the 3D models

There still exists a problem that could occur. Not all the models that are loaded are centered at the origin point. To solve this, a camera will be created, and the models will be moved in space to centralize them. Here is the approach : First a THREE.Box3 object is created. For each STL model that needs to be loaded, the box is extended to include the 3D models. The center of the box is obtained using the getCenter() function of Box3. All the models are moved by subtracting the center of the Box3 from the mesh. Finally, the camera is set to focus on the loaded models. Even if a rotation is applied, the camera will always be fixed on the center of the loaded models. It's worth noting that this approach may encounter problems if the models are located far apart. However, within the context of a human body, the proximity should be sufficient. If any issues persist, users have the option to load models independently. A similar code to the explained approach is the Listing 3.7:

```
//STLS is an array containing the different STL files
const camera = new THREE.PerspectiveCamera(60, 1920/1080, 1.0, 1000.0);

const box = new THREE.Box3();
for (let i = 0; i < STLS.length; i++) {
    box.expandByObject(STLS[i]);
}
const center = box.getCenter(new THREE.Vector3());
```

```
for (let i = 0; i < STLS.length; i++) {
    STLS[i].position.sub(center);
}

camera.position.set(center.x, center.y, box.max.z + 50);
camera.lookAt(center);
```

Listing 3.7: JavaScript code to generalize the creation of a camera for the scene

The rest of the code is not really hard to implement. Three.js has many tutorials on the internet to find which elements have to be added to finish the scene and how to add commands to the keyboards to rotate and zoom on the models.

3.7 Print the models

Once the models can be easily obtained by the codes and we can see the results using different softwares, the next step is to check if the STL files are printable by 3D printers. There are many software and hardware options available as many companies create such engines. In this work, products from Ultimaker Company will be used. The Ultimaker Cura software will help prepare the STL file for printing. The Ultimaker 2+ and Ultimaker 3 will be used to print the file once it has been processed by the software. Ultimaker Cura can easily handle STL files, so no additional processing is required, we simply need to load the files into the software.

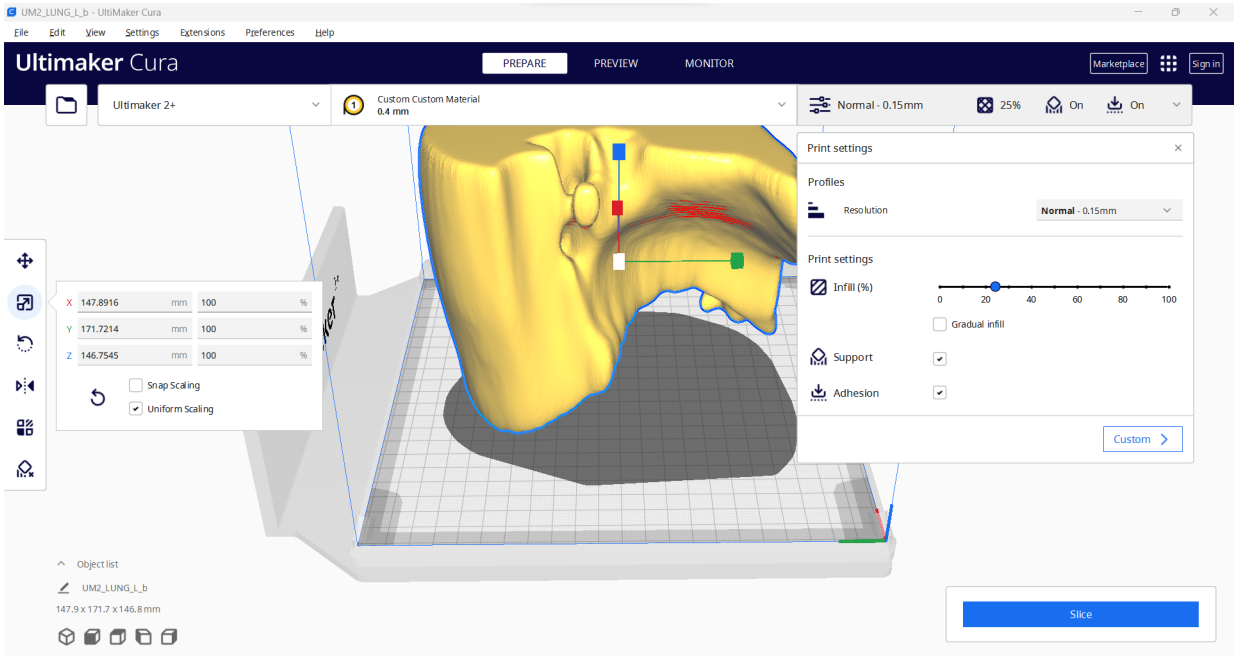


Figure 3.4: View from Ultimaker CURA software

As we can see on Figure 3.4, the software is clear and many options are available. Without any modification, the users can click on *slice* to create the file with the instructions for the printers. But all the models are different and problems could occur. An example is part of the models that are above the void without any support. Here is a list of different settings that could be modified to get good results.

- **Model** : The model of the printers can be changed because they don't have the same size, possibilities...
- **Size** : The size can be changed. It's useful since printing organs at the full size could ask for a lot of time and plastic resources
- **Orientation** : The model can be rotated because sometimes, a rotation will reduce the time needed for the print

- **Support** : If the model is complex with parts with void under it. support can be generated by the software to avoid a collapse during the print
- **Filling** : The users can change the way that the inside of the model is filled. It allows to change the time for the print and the resistance of the model.

Other elements are configurable but those are the main elements that are important to have a good printed model. All those possibilities say that no more work is needed during the creation of 3D models. The marching cubes don't have to be modified because software like Ultimaker Cura will do the rest of the job. So once the configuration is done, the users have to press the slice. Wait to get the instruction file. Put this file into a printer, launch the print and wait until the end.

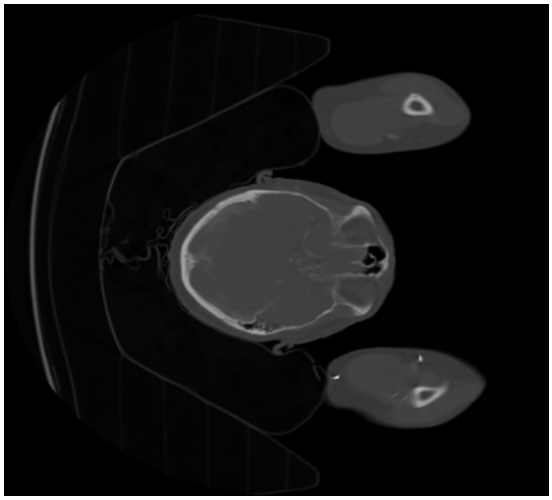
3.8 Use of AI

To simplify the process of creating 3D models from scans, artificial intelligence can be utilized. Based on the knowledges of the state-of-the-art section, TotalSegmentator appears to be a promising tool for this thesis. However, before integrating it into the code, we need to gather more specific details. The project's GitHub page mentions the availability of a Python library for TotalSegmentator, which is freely accessible. This is a positive initial step, considering that the previous work for this thesis has also been done in Python. The API can be called with a lonely function. The declaration of this function can be found in the source file :

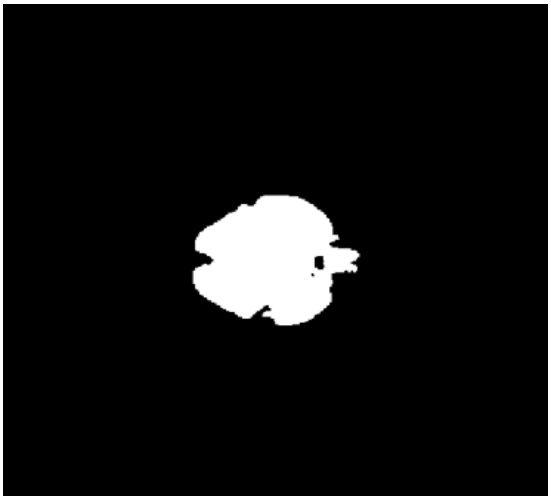
```
def totalsegmentator(input, output, ml=False, nr_thr_resamp=1,
    nr_thr_saving=6, fast=False, nora_tag="None", preview=False,
    task="total", roi_subset=None, statistics=False, radiomics=False,
    crop_path=None, body_seg=False, force_split=False,
    output_type="nifti", quiet=False, verbose=False, test=0):
```

Listing 3.8: Definition of totalsegmentator function in the API Python code

The next crucial aspect pertains to the input and output data. After carefully reading the documentation and utilizing a website developed by the creators for online usage of TotalSegmentator, we discovered the following information. As for the input data, we have two options: a zip file containing a complete series of DICOM CT scans or a NIfTI file containing only the images from the DICOM series. With pydicom in Python and other packages, it's not complicated to manipulate a DICOM CT Scan series to create a correct NIfTI file. The output consists of a series of NIfTI compressed files that contain masks for up to 104 different structures. Each segmented element corresponds to a single file within the series. It is important to note that each output file will contain the same number of masks as the initial series even if some masks are empty. For instance, if the initial series consists of 100 CT images, each output file will also contain 100 masks. The Figure 3.5 is an example of an input and output image illustrating brain detection:



(a) Example of CT image given to TotalSegmentator



(b) Resulting mask corresponding to brain detection

Figure 3.5: Example of input and output data of TotalSegmentator

Those output files are really interesting because it's perfectly the kind of input that we give to the Marching Cube algorithm in the first part of this work. The masks are kind of bitmap so no prepossessing has to be done on data. It's also interesting because we don't need to manage spatial information since we get only bitmap and no contours like in RT Struct files and the position relative to the input data can be obtained by the index of a pixel. Also, the input images and the resulting masks have the same size so there is no distortion that has to be managed. The only element that has to be found is the distance between two scans on the original series.

However, it still exists a small problem that is already noticed and which could be managed easily. Indeed, as described before, the NIfTI results contain always the same number of images for a same series as input, and because some of the masks could be empty, the computation time of the Marching cubes algorithm will explode if it has to work on all bitmaps. A solution is to add a test before the iteration of two new bitmaps. If the two bitmaps are empty, no triangles will be created between them. So the process can pass this part and continue with a new tranche created with the following bitmap. This check can be done easily with some function integrated in python as shown the Listing 3.9 example. The function .any() will check if in the bitmap, there exists at least one pixel which is set to 1. the condition will return false if two bitmap are empty:

```
    for L in range(len(bitmap)-1):
        if(bitmap[L].any() or bitmap[L+1].any()):
            for i in range(len(bitmap[0])-1):
                for j in range(len(bitmap[0][0])-1):
```

Listing 3.9: Python code in the marching cubes to check if two bitmap are empty or not

So the process to recreate 3D models from Dicom CT scan series is easier than from RT Struct files. The still exist an interrogation about the quality of the result models. AI could make errors or some small elements could be fusion due to errors of detection

3.9 User Interface

To simplify the usage of the codes for potential users, a user interface can be built. The Tkinter library in Python allows the creation of such windows. The process is not difficult. We just need to create buttons, text zones, and a way to select files in a file explorer. However, it can be challenging to consider all the edge cases that may arise. This UI needs to be simple enough for everyone to use, including those who are not frequent computer users. The software needs to be simple to launch and the Python libraries have to be easily installed.

For this final point, a simple check can be performed before executing the code. Specifically, the code can examine the installed libraries on the computer. If a required library is not present, the code can install it using pip. If all the libraries are already installed, the code proceeds with its execution.

One important aspect to consider is that the Python code may be located in different folders. To avoid import problems during code execution, the paths to those folders need to be added to the Python path. This problem can be simplified by using a .bat file in Windows or a .sh file in Linux. If these files contain the correct commands, the user will only need to click or double-click on the file to execute the software. The code in the Listing 3.10 is an example of such an executable file, including the addition of the Python path :

```
setlocal
REM path to folder where is locate the scrypt
set "SCRIPT_PATH=%~dp0"
REM relative path to utils and App
set "UTILS_PATH="%SCRIPT_PATH%utils""
set "APP_PATH="%SCRIPT_PATH%App""
REM Add folder to PYTHONPATH
set "PYTHONPATH=%UTILS_PATH%;%APP_PATH%;%PYTHONPATH%"
REM Execution of Python code
start "" python "%SCRIPT_PATH%App\Main.py"
endlocal
```

Listing 3.10: Bash code to add temporarily paths to Python path

Chapter 4

Results

This section will reuse the similar path as the methodology chapter, implementing each design choice that was made to produce the results presented here. By displaying the results, it will be possible to determine whether the choices were effective or not, and whether any previously known disadvantages are still present. Each result will be presented individually to derive conclusions about the works. Some results will be completed by a link to a youtube video. Many results of this thesis are more understandable in a video than in pictures. Video will sometimes help to show that everything works correctly.

To demonstrate the real-world applicability of the source code, The Cancer Imaging Archive (TCIA) was used to retrieve DICOM RT Struct files and DICOM CT scan series at no cost. All files were anonymized prior to uploading to the archive, eliminating the need to erase any data related to specific individuals. However, it should be noted that this does not hold true if the work produced for this thesis is used by malicious individuals. As we will see later in this chapter, certain 3D reconstructions can be easily obtained, and using them for non-medical purposes is not unthinkable.

4.1 Results of Marching Cubes algorithm

It could be difficult to represent a 3D object with 2D images. All the results are screenshot from models displayed with the software 3D Builder from Microsoft to try to show them as interesting as possible. By taking a random file from TCIA, we can obtain the following results in Figures 4.1-4.3 by using the normal version of the Marching Cube Algorithm. Each model was created with bitmaps of size 200. The name which is written below each image is the name given to a set of contours representing 3D models. Even if for some models it's difficult to understand which part of the body it is, the others like LUNG_L and LUNG_R are recognizable.

On the good aspect of the marching cubes, we can see from the different results that no branching leads to problems in the 3D models and that the models seem to be correctly built. The method can also manage problems in the original data. As we see in the bottom right of the Bowel_sm_planCT, there is a small part that is not connected to the rest of the model. The origin of the problem seems to be in the original data because if the problem comes from the codes, we would see problems on others models which is not the case.

Even if all the models are closed and there is not hole allowing to see the inside of each model, the results are not smooth. This problem was already identified but it is more concrete when results are displayed. A way to solve the problem a bit is by reducing the size of the bitmap but this is not a good way. previously, the choice was made to create a bitmap with a size of 200. After plotting the results for the marching cubes, four 3D models of the LUNG_L model are plot in Figure 4.4 to show that even if the sizes of the bitmap are modified, the results don't change a lot with those for a 200 size and keep to have this blocky aspect.

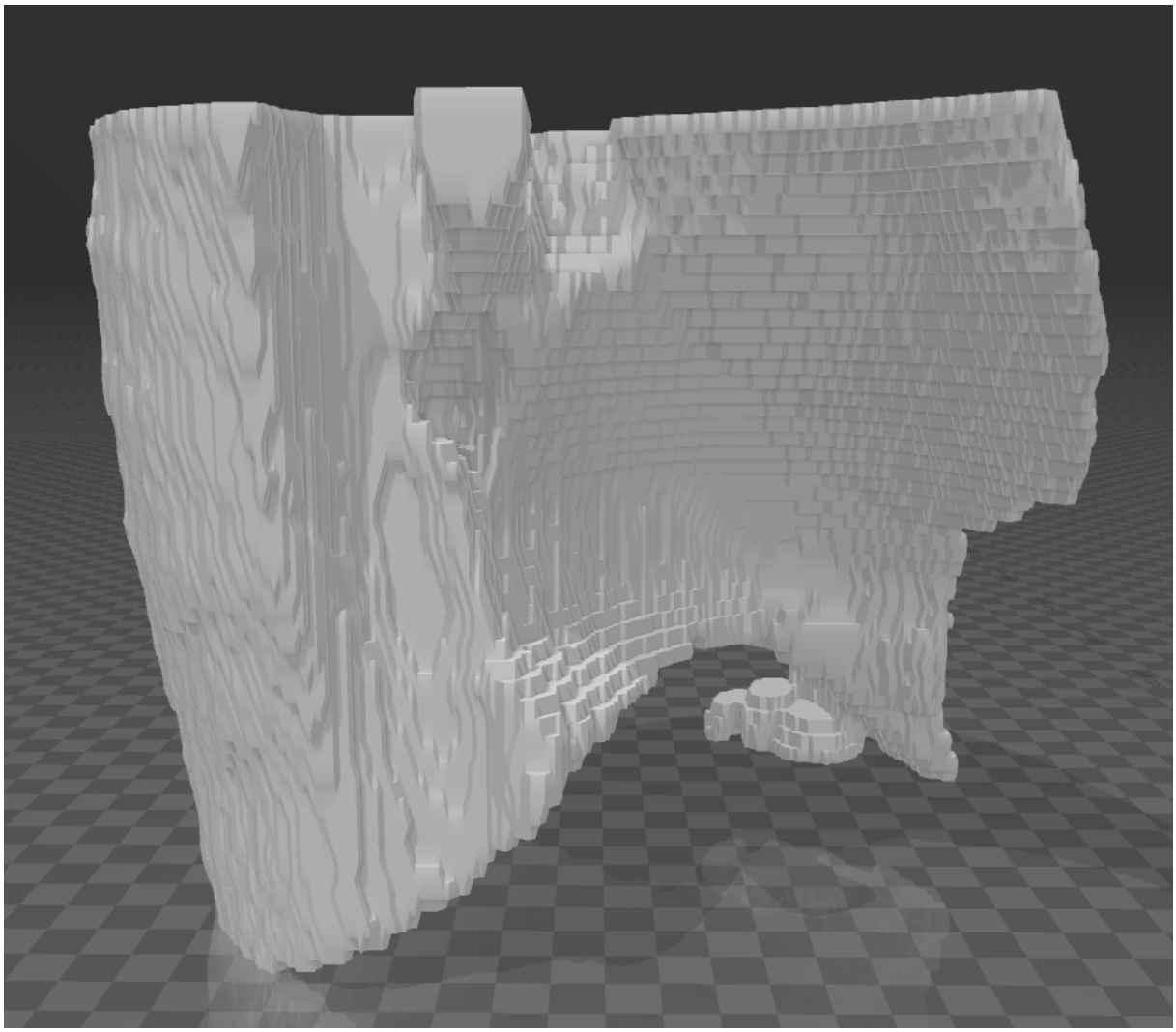
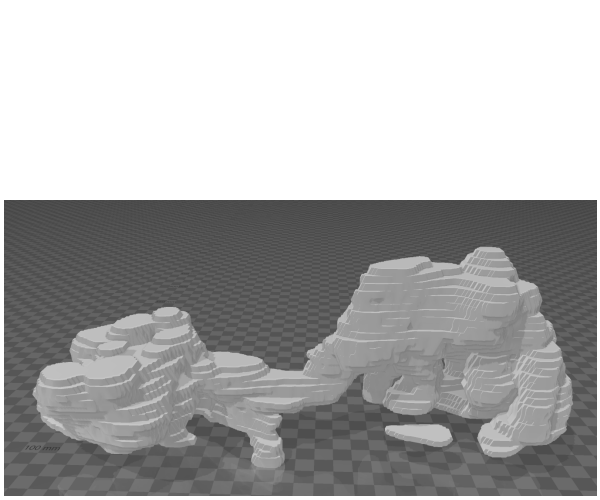
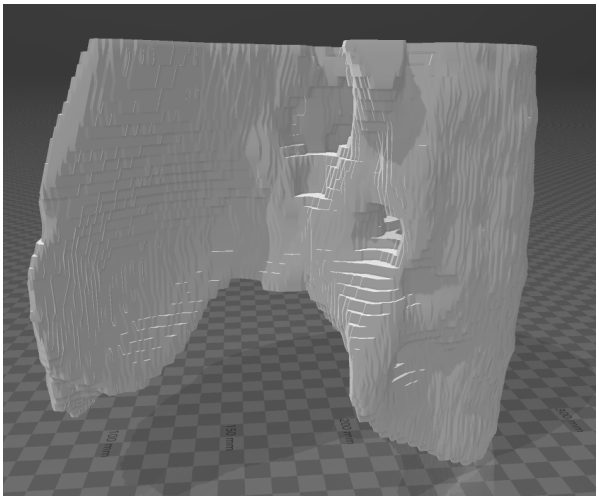


Figure 4.1: Model Name : LUNG_L

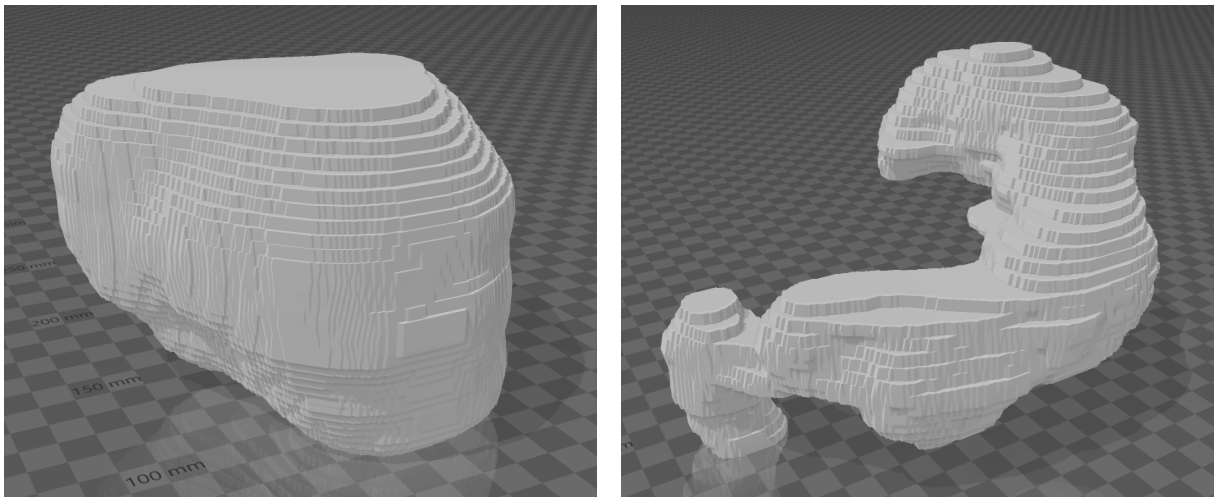


(a) Model Name : Bowel_sm_planCT



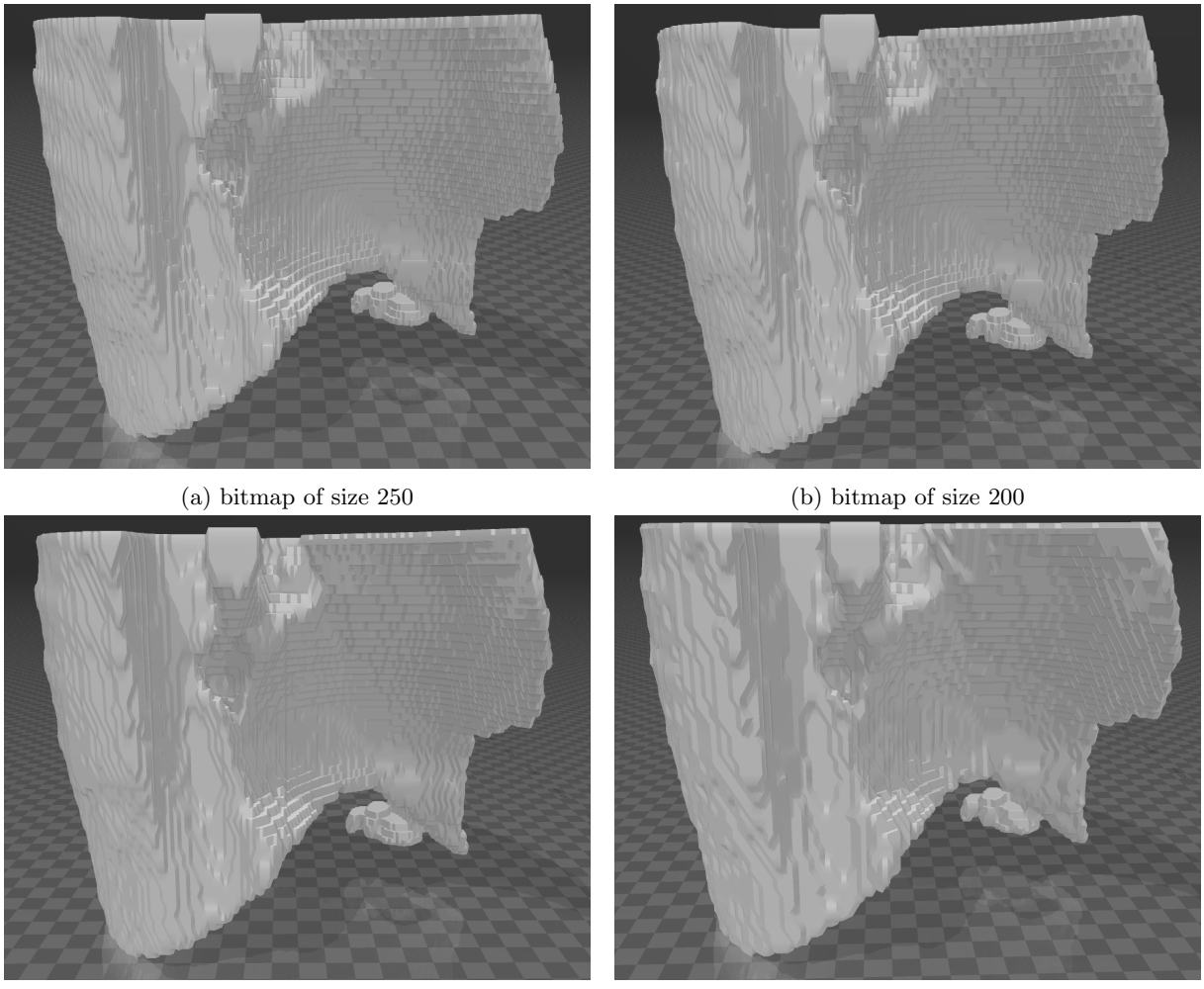
(b) Model Name : LUNG_R

Figure 4.2: Set of 3D models



(a) Model Name : ROI (b) Model Name : Stomach_duo_planCT

Figure 4.3: Set of 3D models



(a) bitmap of size 250 (b) bitmap of size 200

(c) bitmap of size 150 (d) bitmap of size 100

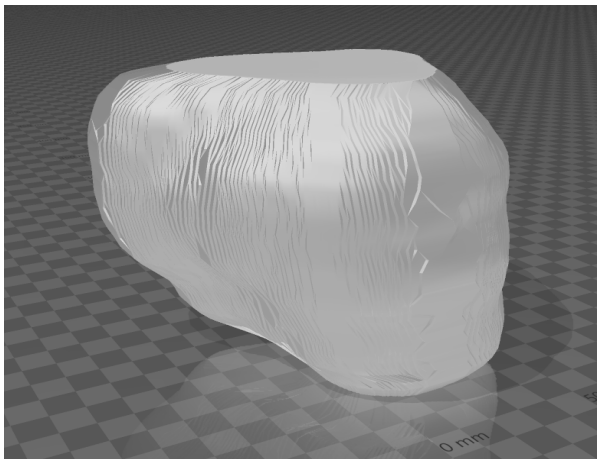
Figure 4.4: Set of LUNG_L model created with differents size for the bitmap

Determining whether the results are good or not could pose a challenge. The input data consists solely of points, which makes it difficult to gauge the accuracy. One approach to assess the fit of the resulting models within the original contours is by displaying the original contours alongside. However, this method may lead to the loss of certain details.

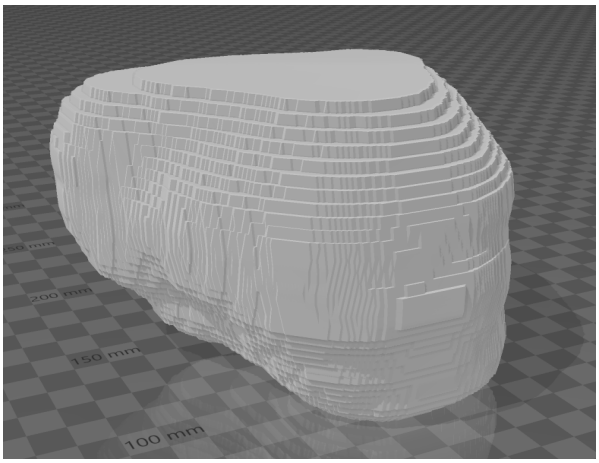
Alternatively, another idea is to incorporate an element introduced in the methodology chapter. In that chapter, incomplete results were presented from an algorithm that was unable to handle problems such as branching. By comparing the outcomes of this imperfect method with the results obtained using the marching cubes technique, certain conclusions can be drawn. If the results differ significantly, it indicates a potential error in at least one of the methods. On the other hand, if the models are similar and concentric, it provides more confidence in the correctness of both methods. This is because they yield comparable outcomes despite being fundamentally different approaches.

Indeed, the incomplete algorithm creates a sort of "headband" by utilizing the points of the contours. One advantage of this method is that it operates directly on the actual points, eliminating the need for any transformations. The algorithm generates models at the same locations as the original data, ensuring that the resulting models maintain the correct size by utilizing the original points. However, these models only provide a general representation of the object's shape. Some parts may be inaccurate or incomplete, as demonstrated in Figure 4.5c.

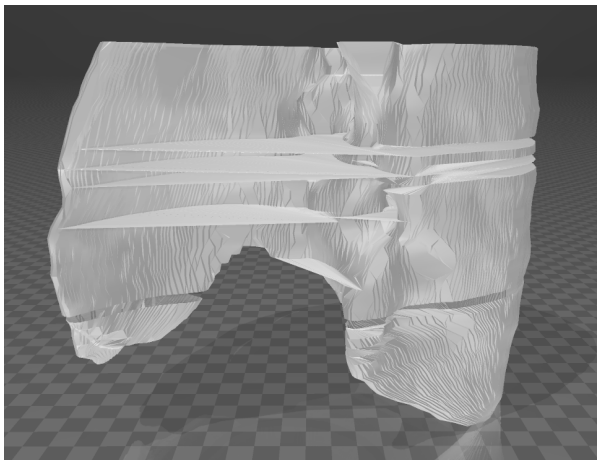
In Figure 4.5, Images of models generated by the incomplete method are displayed and are accompanied by the corresponding results obtained through the marching cubes technique. The visual similarity between the two is evident. To further support this observation, a video demonstrating the size consistency and spatial alignment of the models can be accessed via the link provided below the images.



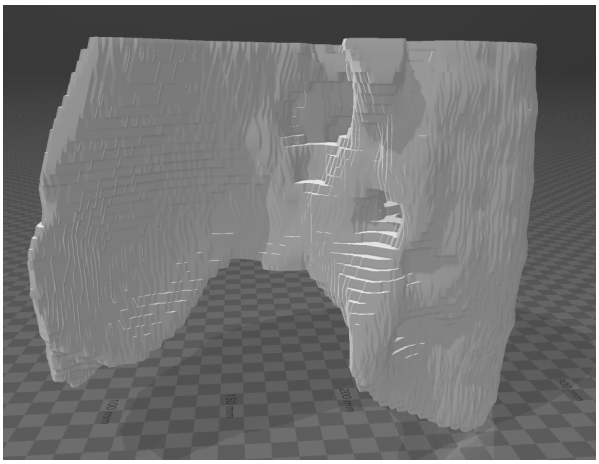
(a) Model : ROI from incomplete algorithm



(b) Model : ROI from Marching cubes



(c) Model : LUNG_R from incomplete algorithm



(d) Model : LUNG_R from Marching cubes

Figure 4.5: Set of 3D models from different algorithms

`https://youtu.be/aW-RSnqSzL0`

4.2 Improved Results of the Marching Cubes Algorithm

4.2.1 Results for Marching Cubes with interpolation

Before presenting the different results of this section, it is important to address a key factor. The interpolation is performed on all edges of the simulated cube. However, in the case of a point being positioned on a vertical edge, this interpolation needs to be disabled. Failure to do so would result in the 3D models exhibiting a lizard skin effect. To illustrate this issue, the zoomed-in sections of models are displayed in the Figure 4.6.

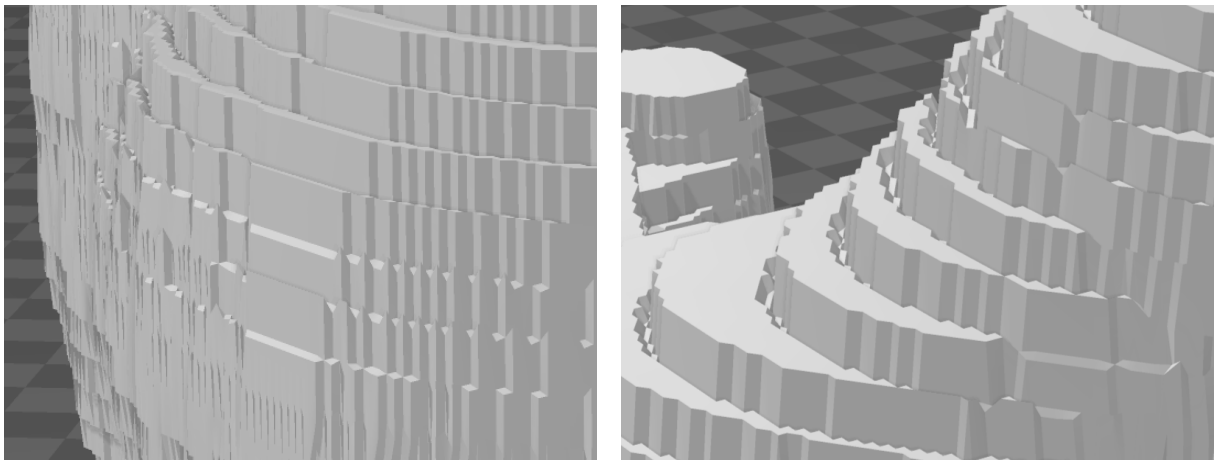


Figure 4.6: Problem on models with interpolation

Once the interpolation on the vertical edge is disabled, the results can be analyzed to evaluate the effectiveness of the method. Three different methods for creating density bitmaps will be tested and compared to the original method. The first method involves taking the mean value of a 3D neighborhood of 8 pixels in two consecutive bitmaps, the second method takes the mean value of a 2D neighborhood of 9 pixels, and the last method applies a Gaussian convolution to the original matrix. The Bowel_sm_planCT models will be utilized to compare the specific shape of the model. Observing Figures 4.7-4.10, it becomes apparent that convolution is the only method that truly impacts the reconstruction.

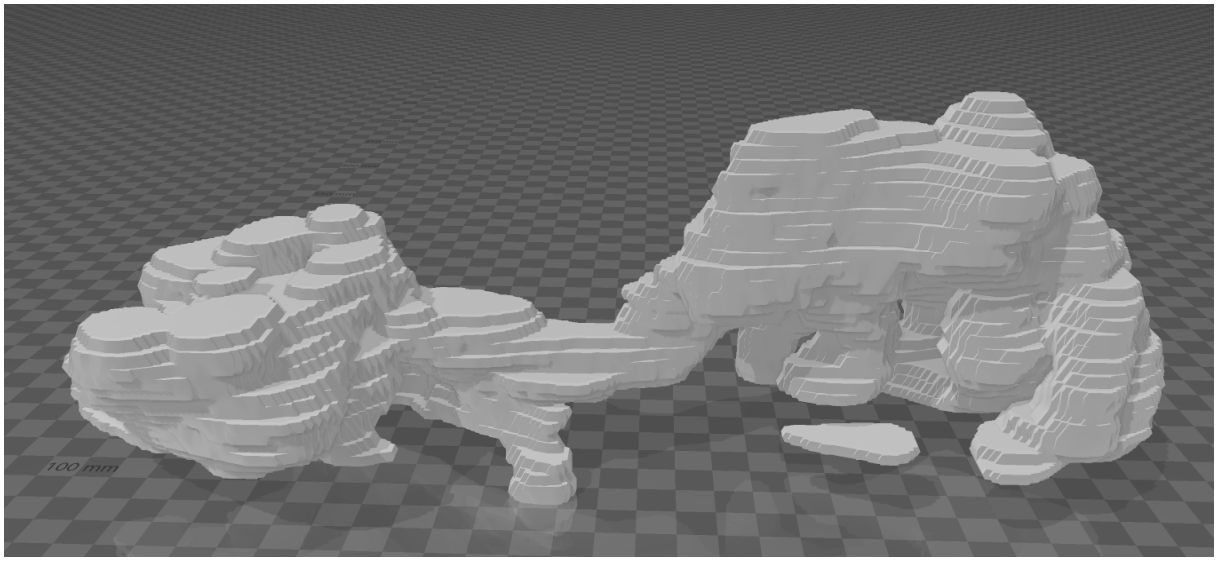


Figure 4.7: Original model

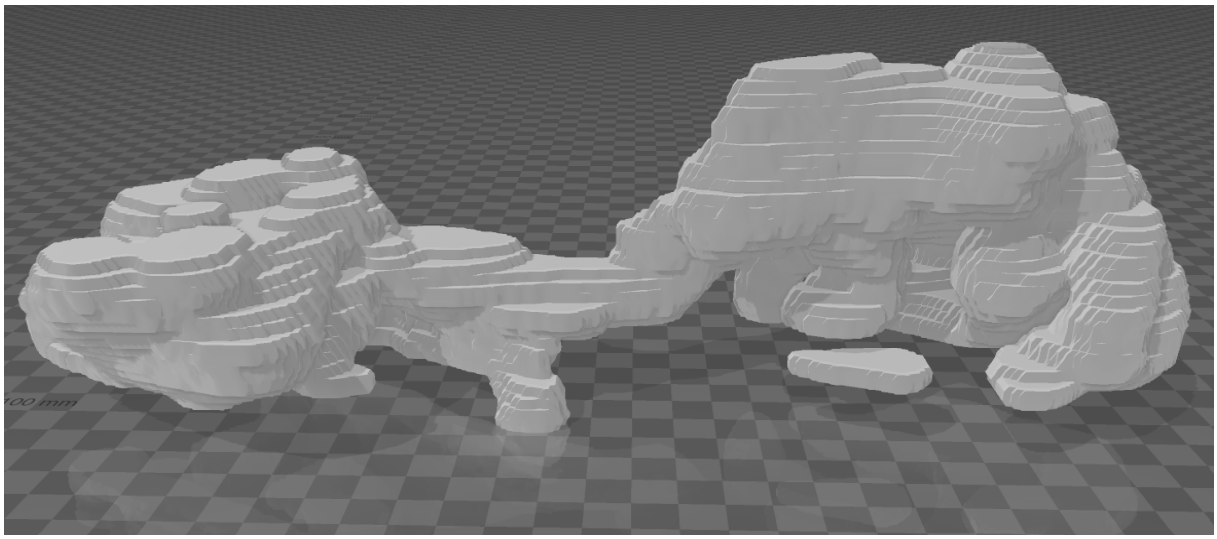


Figure 4.8: Model created with a 3D neighborhood of 8 pixels

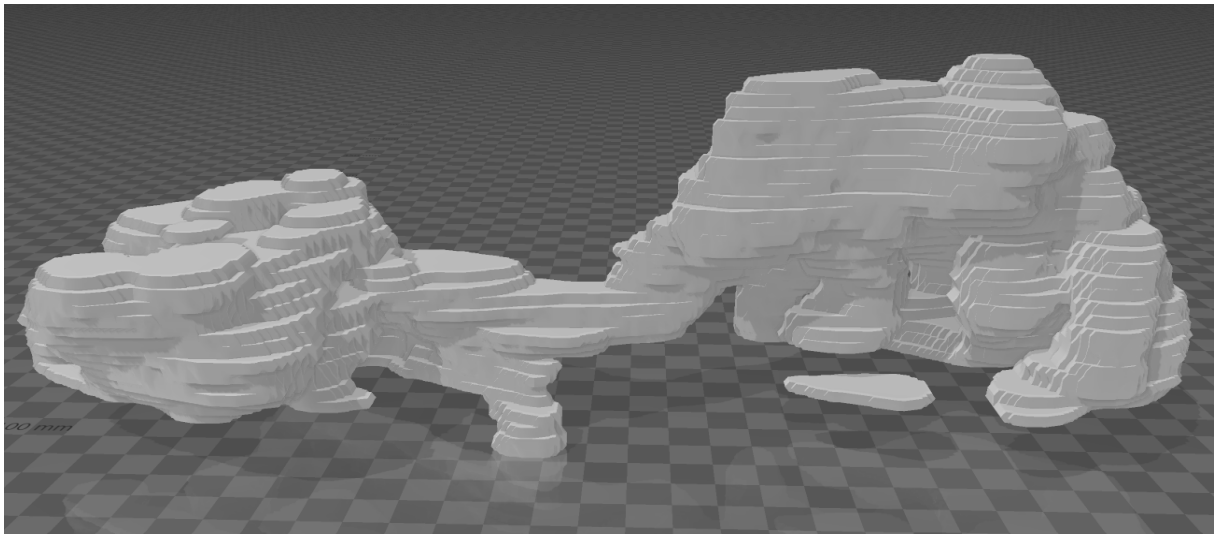


Figure 4.9: Model created with a 2D neighborhood of 9 pixels

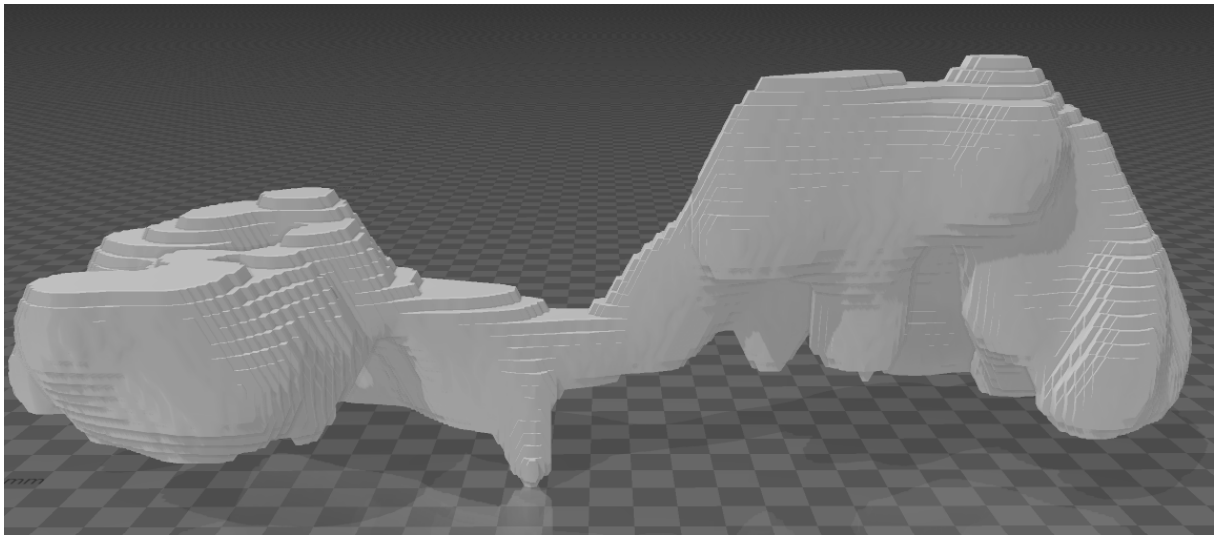
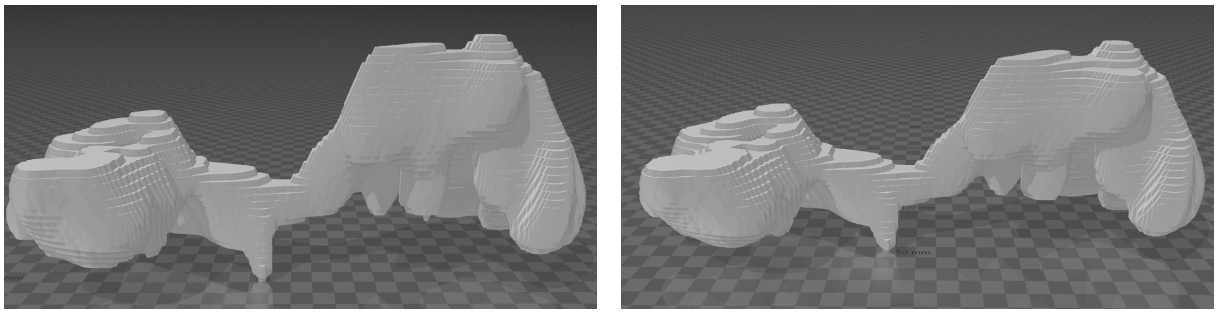


Figure 4.10: Model created with a Gaussian convolution

The difference between the original model and the "improved" version is not significant. As described in the methodology chapter, this may be attributed to the small size of each simulated cube in the Marching Cubes algorithm, limiting the point's ability to move extensively. Consequently, while there is a slight improvement in the results, the main issue of the stair-like appearance still persists. Only the convolution method remains noteworthy as it effectively removes impurities and holes present in the original data. The resulting models appear slightly smoother, but when the density matrix obtained from the convolution is applied to the normal version of the marching cubes algorithm, the results are nearly equivalent, albeit with a more streamlined approach.

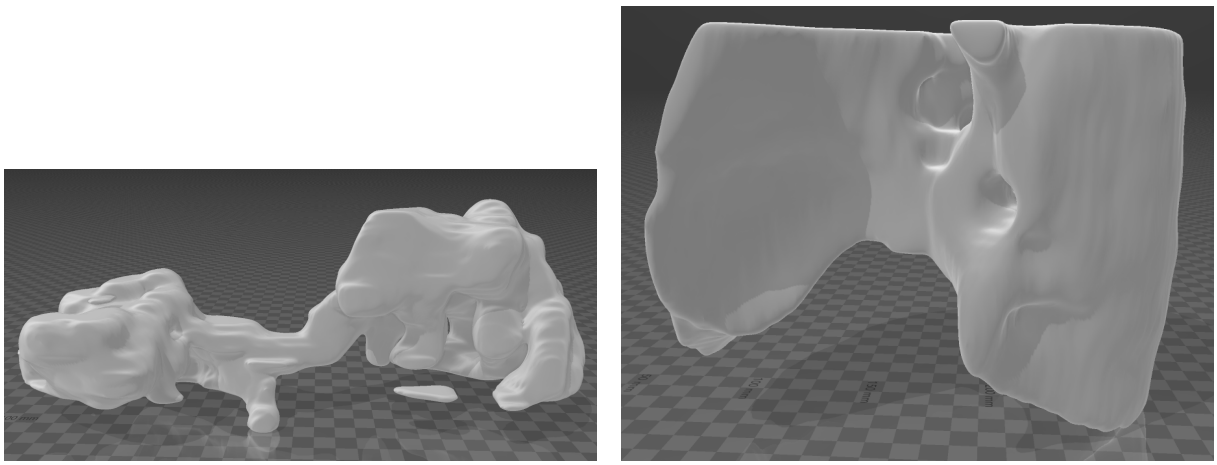


(a) With interpolation (b) Without interpolation

Figure 4.11: Comparison of results using gaussian convolution with and without interpolation

4.2.2 Results for post-processing the meshes

If we apply the algorithm from the VTK library, we can obtain smoother results, as shown in the Figure 4.12. The model no longer exhibits a staircase-like appearance, and instead appears more representative of real organs. However, it is important to compare the volume of the model after smoothing with VTK to the volume of the original version. The results of this comparison for the five different models used in the previous section are displayed below the images. The last column indicates the percentage of difference by considering the volume of the original model as a reference. It can be observed that the volumes are similar, indicating that this smoothing method does not significantly reduce the model's volume. Figure 4.12 is the result of the operation of the algorithm coming from VTK, as it changes the visual appearance of the model without changing its size.



(a) Model Name smoothed : Bowel_sm_planCT (b) Model Name smoothed : LUNG_R

Figure 4.12: Smoothed models with VTK library

Name of the element	Volume : original version	Volume : smoothed version	difference (%)
Bowel_sm_planCT	429647 (mm^3)	420549 (mm^3)	2,12
LUNG_L	1353668 (mm^3)	1349269 (mm^3)	0,32
LUNG_R	1365952 (mm^3)	1360917 (mm^3)	0,37
ROI	630446 (mm^3)	627848 (mm^3)	0,41
Stomach_duo_planCT	271977 (mm^3)	268255 (mm^3)	1,37

Table 4.1: Compared volume of a original version and a smoothed version

4.2.3 Results by combining the two methods

By utilizing both techniques to enhance the quality of the results and applying them to the five original models presented in the section Results of the Marching Cubes algorithm, we can achieve smoothed models that are free from impurities.

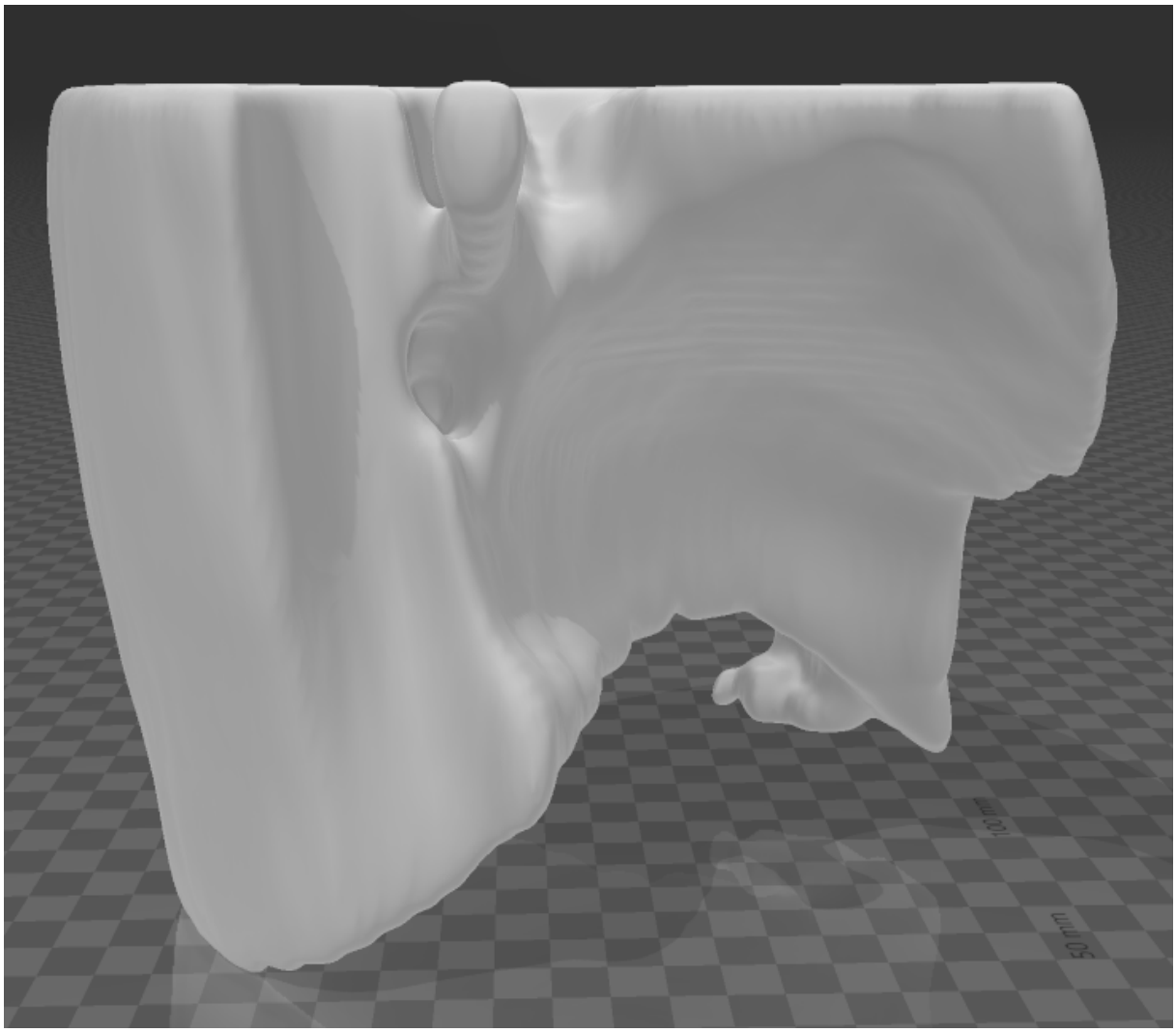
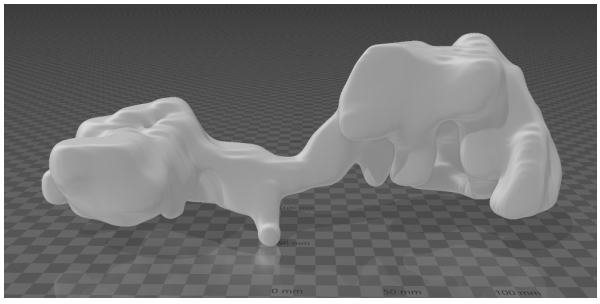
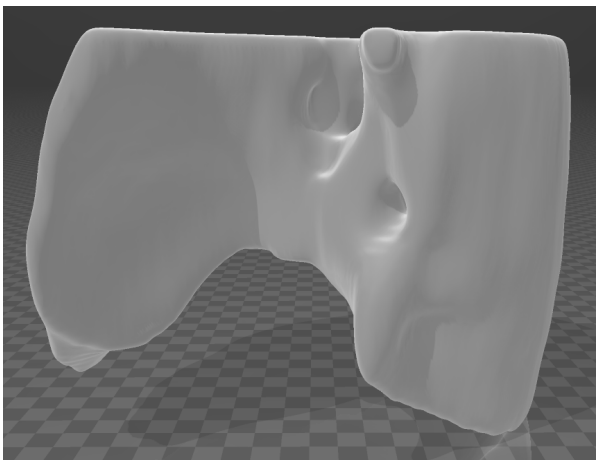


Figure 4.13: Model Name : LUNG_L

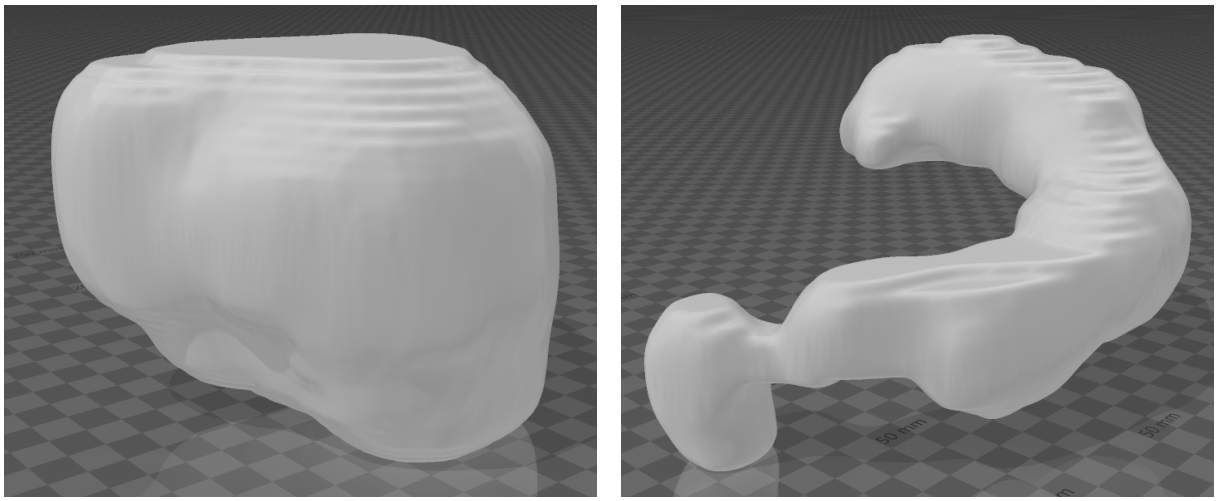


(a) Model Name : Bowel_sm_planCT



(b) Model Name : LUNG_R

Figure 4.14: Set of 3D models



(a) Model Name : ROI (b) Model Name : Stomach_duo_planCT

Figure 4.15: Set of 3D models

4.3 Creation of DICOM STL Encapsulated

As explained in the methodology chapter, it is quite simple to test whether a DICOM file is correct or not. Using the dicom3tools library in a terminal is straightforward, as we only need to provide the software with the path to the DICOM file. However, before displaying the results of the code, the results for a deliberately incorrect DICOM file will be shown in Figure 4.16 to illustrate the type of error messages obtained in case of a problem.

```
Windows PowerShell
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Documents\QX Memoire\Python
Code Version\App\test_error.dcm"
Error - MediaStorageSOPInstanceUID but missing SOPInstanceUID and not a directory
Error - MediaStorageSOPClassUID but missing SOPClassUID and not a directory
Warning - Missing attribute or value that would be needed to build DICOMDIR - Patient ID
Warning - Missing attribute or value that would be needed to build DICOMDIR - Study Date
Warning - Missing attribute or value that would be needed to build DICOMDIR - Study Time
Warning - Missing attribute or value that would be needed to build DICOMDIR - Study ID
Warning - Missing attribute or value that would be needed to build DICOMDIR - Modality
Warning - Missing attribute or value that would be needed to build DICOMDIR - Series Number
Error - Information Object Not found
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools>
```

Figure 4.16: Results from dciodvfy with a wrong DICOM file

```
Windows PowerShell
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Desktop\check_dciodvfy\Bowel
_sm_planCT.dcm"
EncapsulatedSTL
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Desktop\check_dciodvfy\LUNG_
L.dcm"
EncapsulatedSTL
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Desktop\check_dciodvfy\LUNG_
R.dcm"
EncapsulatedSTL
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Desktop\check_dciodvfy\ROI.d
cm"
EncapsulatedSTL
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools> .\dciodvfy.exe "C:\Users\Thibault Gregoir\Desktop\check_dciodvfy\Stoma
ch_duo_planCT.dcm"
EncapsulatedSTL
PS C:\Users\Thibault Gregoir\Desktop\dicom3tools>
```

Figure 4.17: Results from dciodvfy with correct DICOM files

We can observe in the Figure 4.17 that none of the created DICOM files presented any issues. However, there was a problem that was discovered afterwards, which is not visible in the picture. But before delving into the details of the problem, it is necessary to provide some context to understand why it was somewhat challenging to identify.

In this work and in the different code implementations, there are two ways to create a DICOM STL Encapsulated file. The first method involves creating a file from scratch, which is useful when we want to add medical information to an independent 3D model. The second option is to use a DICOM file as a template. If the codes create 3D models from a DICOM RT Struct file, we can reuse this DICOM file by removing unnecessary tags and retaining relevant information from the original data. Therefore, the basis of this version is a DICOM file that has already passed some checks because it was created by another software.[1]

The main difference between these two solutions is the absence of the **DICM** prefix in the "scratch" version. This absence causes problems with the DICOM parser, which raises an error when trying to read these DICOM files. The difficulty in identifying this problem was compounded by the fact that it was only present in one of the methods. Part of the error message raised by pydicom is displayed in Listing 4.1:

```
>>> import pydicom as pm
>>> pm.dcmread("ROI.dcm")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
...
pydicom.errors.InvalidDicomError: File is missing DICOM File Meta Informatio
header or the 'DICM' prefix is missing from the header.
Use force=True to force reading.
```

Listing 4.1: Error given by DICOM parser if no DICM predix is present

This problem can easily be solved by adding those three lines of codes if the user wants to create a DICOM STL encapsulated from scratch. An example of resulting file is display in Listing 4.3 :

```
DS = pydicom.Dataset()
preamble = np.zeros(128, dtype=np.uint8) #Added lines
preamble[:4] = [68,73,67,77]             #Added lines
DS.preamble = preamble.tobytes()         #Added lines
```

Listing 4.2: Python code to solve the previous problem

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length UL: 208
(0002, 0001) File Meta Information Version      OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID        UI: Encapsulated STL Storage
(0002, 0003) Media Storage SOP Instance UID     UI: 1.2.826.0.1.3680043.8...
(0002, 0010) Transfer Syntax UID                UI: Implicit VR Little Endian
(0002, 0012) Implementation Class UID           UI: 1.3.6.1.4.1.22213.1.143
(0002, 0013) Implementation Version Name        SH: '0.5'
-------------------------------------------------
(0008, 0016) SOP Class UID                       UI: Encapsulated STL Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.826.0.1.3680043.8...
(0008, 0020) Study Date                          DA: '20230526'
(0008, 0023) Content Date                        DA: '20230526'
(0008, 002a) Acquisition DateTime                DT: '20230526110750.675404'
(0008, 0030) Study Time                          TM: '110750.674404'
(0008, 0033) Content Time                        TM: '110750.675404'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'M3D'
(0008, 0070) Manufacturer                        LO: 'Thibault Gregoir'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 1090) Manufacturer's Model Name           LO: 'Master Thesis Programs'
(0010, 0010) Patient's Name                      PN: ''
```

[1] If errors occur with data that was not generated by the code in this work, the problem originates from elsewhere that we can't really control.

```
(0010, 0020) Patient ID                                  LO: 'TG711765'
(0010, 0030) Patient's Birth Date                        DA: '20000101'
(0010, 0040) Patient's Sex                               CS: 'M'
(0018, 1000) Device Serial Number                        LO: '753951'
(0018, 1020) Software Versions                           LO: '2'
(0020, 000d) Study Instance UID                          UI: 1.2.826.0.1.3680043.8...
(0020, 000e) Series Instance UID                         UI: 1.2.826.0.1.3680043.8...
(0020, 0010) Study ID                                    SH: '753951'
(0020, 0011) Series Number                               IS: '1'
(0020, 0013) Instance Number                             IS: '1'
(0020, 0052) Frame of Reference UID                      UI: 1.2.826.0.1.3680043.8...
(0020, 1040) Position Reference Indicator                LO: ''
(0028, 0301) Burned In Annotation                        CS: 'NO'
(0040, 08ea)  Measurement Units Code Sequence  1 item(s) ----
   (0008, 0100) Code Value                                  SH: 'mm'
   (0008, 0102) Coding Scheme Designator                    SH: 'UCUM'
   (0008, 0103) Coding Scheme Version                       SH: '1.5'
   (0008, 0104) Code Meaning                                LO: 'STL file constructed'
   ---------
(0040, a043)  Concept Name Code Sequence  0 item(s) ----
(0042, 0010) Document Title                              ST: 'Bowel_sm_planCT'
(0042, 0011) Encapsulated Document                       OB: Array of 2983484 elements
(0042, 0012) MIME Type of Encapsulated Document          LO: 'model/stl'
```

Listing 4.3: Exemple of content of a created DICOM STL Encapsulated

We observed in the previous test that the DICOM files are constructed correctly. However, there is still a question mark regarding the proper encapsulation of the STL files. There are two ways to demonstrate this. The first method involves encapsulating an STL file into a DICOM file and then extracting it to check if the extracted file can be displayed in a 3D viewer. The second method will be presented in the *Web Viewer* section of this chapter. If different technologies can retrieve the 3D models from a DICOM file, our approach can be considered as correct. The link below leads to a video demonstrating the results of the encapsulation-extraction process of an STL file, showing its compatibility with the 3D Viewer from Microsoft.

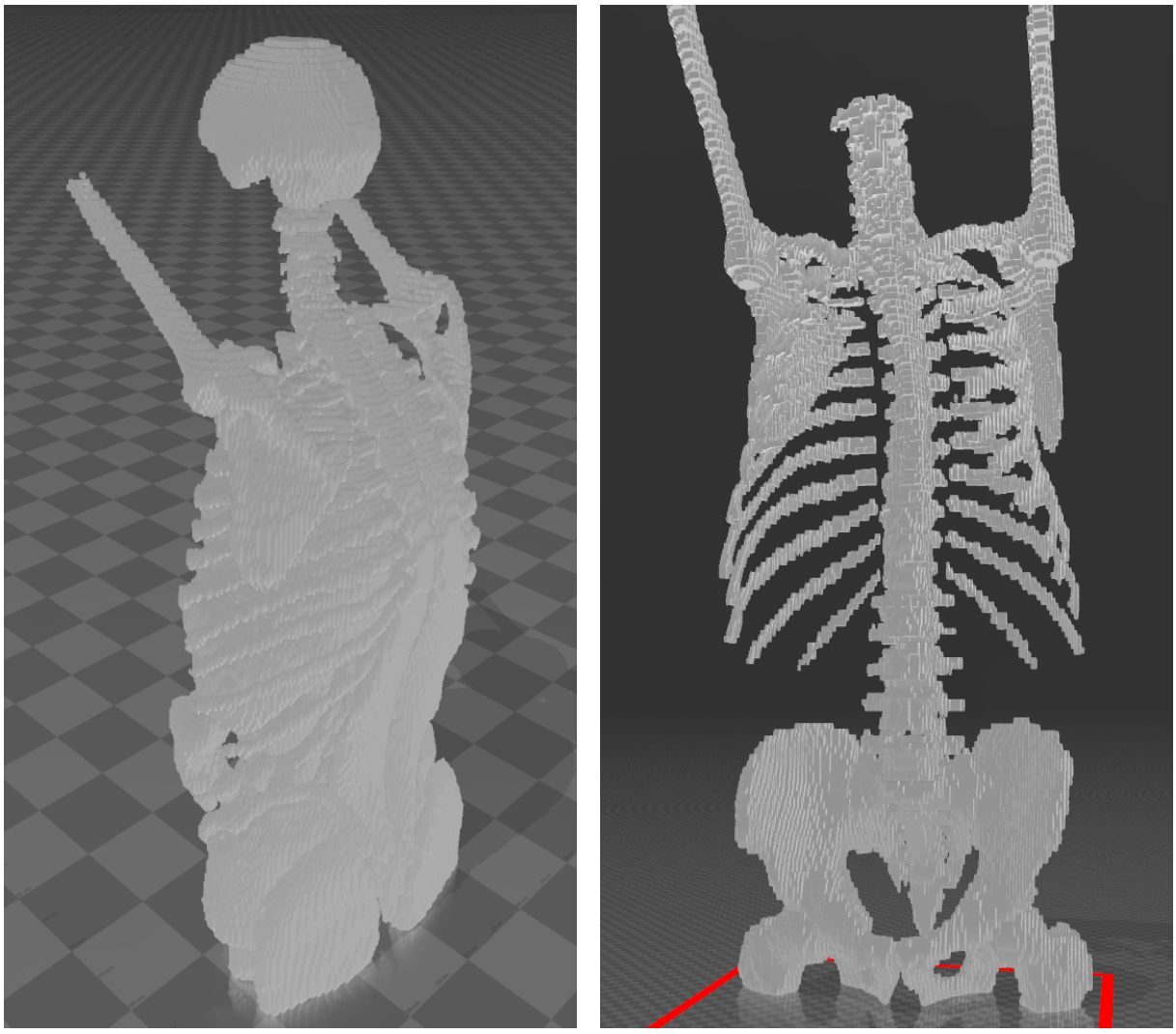
`https://youtu.be/hAFxIJgPS80`

4.4 Use of AI

Since TotalSegmentator is a large and new tool, it may encounter certain issues. Specifically, the tool may suffer from bugs and require significant computational resources. The tests conducted to obtain the models in this section were performed on a computer located in the INTEL room at UCLouvain. The computer had a 16-core CPU and 16 GB of RAM. When executing TotalSegmentator on this system, the artificial intelligence (AI) utilized approximately 50% of the CPU's computational resources and at least 6 GB of RAM.

Despite the computer being powerful enough, it was impossible to segment a complete series of CT scans (around 175 scans) with the 'fast' parameter set to false. However, when the 'fast' parameter was set to true, segmentation became possible. When the parameter was set to false, we had to provide fewer scans for segmentation (around 120) to achieve the segmentation. When requesting segmentation of the body's skin, the AI correctly segmented the skin but did not provide the file containing the mask. These issues are evidently related to the AI, and only the providers can address them.

The following results were obtained by executing TotalSegmentator on a series of CT scans and applying the Marching Cubes algorithm to the outputs. Some results will appear smooth, while others may not. It is important to note that these results are presented solely to showcase the potential of AI or to demonstrate the combination of multiple models in a 3D render. As there are no new algorithmic parts introduced in this part, the results are purely for display purposes.



(a) 103 3D models combined (b) Skeleton

Figure 4.18: Set of 3D models segmented by TotalSegmentator

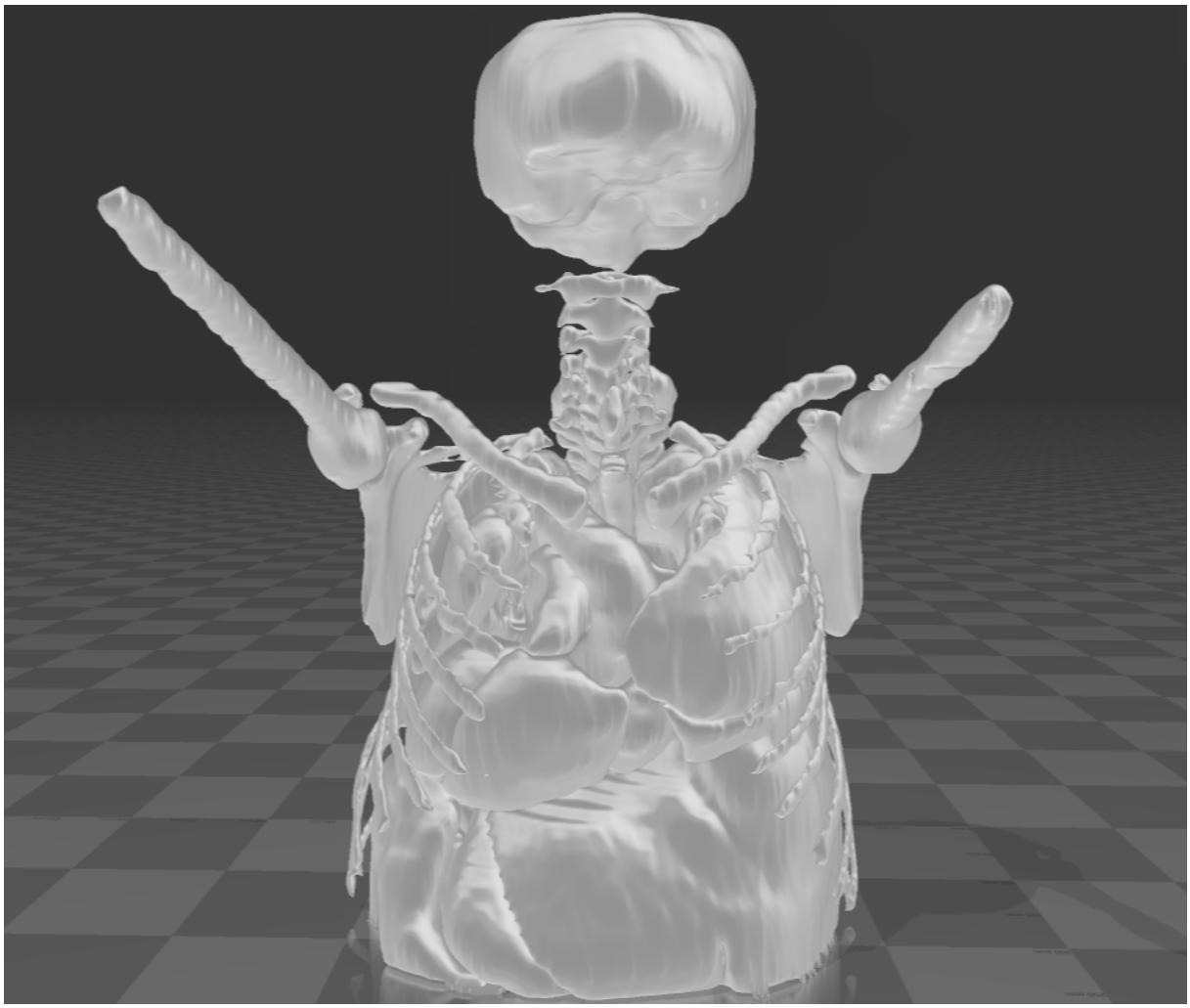
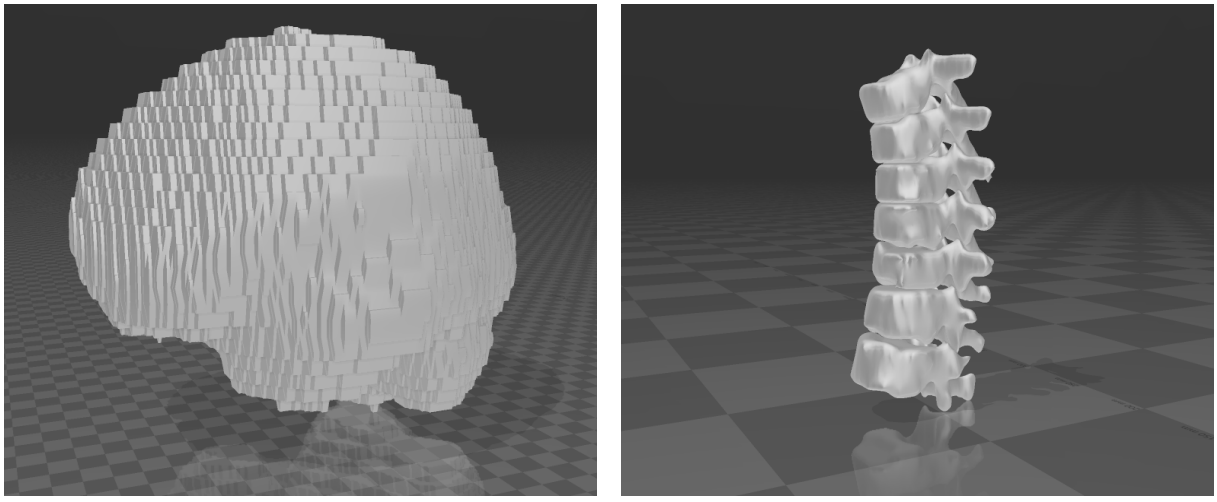


Figure 4.19: Combined and smoothed 3D models

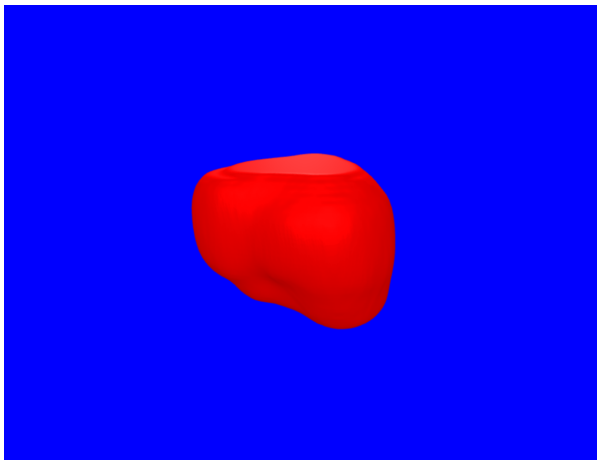


(a) Brain (b) Vertebrae T6-T12

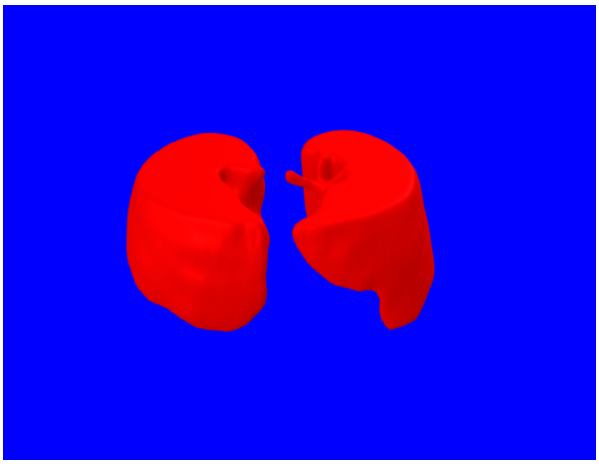
Figure 4.20: Set of 3D models segmented by TotalSegmentator

4.5 Web Viewer

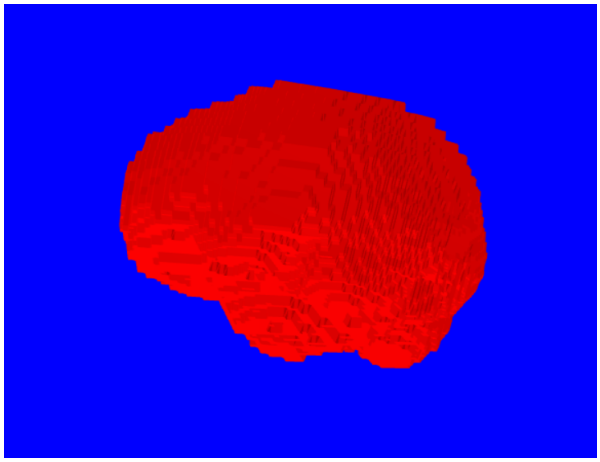
This section may be difficult to illustrate with images, as the primary requirement for the web viewer is to accurately display one or more models. Successful display of the models indicates that the parsing has been performed correctly. The images in Figure 4.21 demonstrate different scenarios where the web application is used to render models, either by performing parsing in Python or directly in JavaScript. To better observe the functionality of this part of the software, accompanying YouTube video have been provided via link.



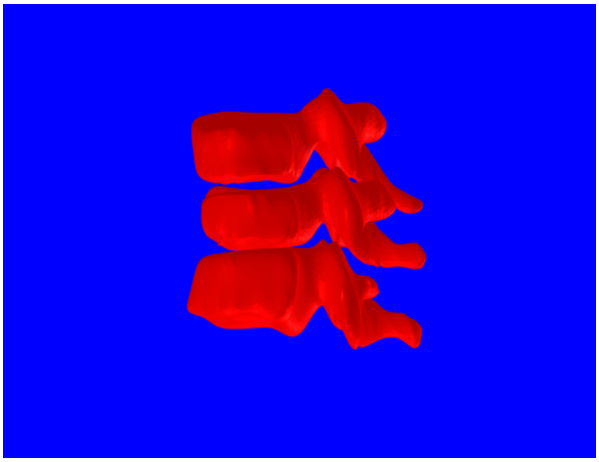
(a) One model parsed in Python



(b) Two models parsed in Python



(c) One model parsed in JavaScript



(d) Three models parsed in JavaScript

Figure 4.21: Set of 3D models displayed in the web application

```
https://youtu.be/gEFLy-TyJbI
```

4.6 Printed Results

With Ultimaker Cura, it is easy to print all the STL files generated by the codes. As described in the methodology section, certain parameters were modified to achieve the desired results. Support structures were generated, and the size of some models was reduced. However, the focus of this section is not on these modifications. The purpose of showcasing the prints is to demonstrate that the models are accurate enough to be processed by printer software. Additionally, it implicitly highlights the ability to recreate body features by combining multiple prints together. By applying the same scaling factor to each print, the size becomes irrelevant as long as the proportions are maintained. Reducing the size also helps in reducing the printing time. The printers used were Ultimaker 2+ and Ultimaker 3, but there were no significant differences between the two models to recommend one over the other.

When the first models were finished printing, measurements were taken to compare the size of the printed objects with their digital counterparts. In general, the sizes were found to be nearly identical. However, a slight deviation of approximately 1mm between the physical and digital versions was commonly observed. This difference can be attributed to the fact that the triangles in an STL file have no physical thickness, whereas the plastic filament used in 3D printers has a specific diameter. As a result, this disparity can introduce a minor error in relation to the 3D model. Since this small element was the only noticeable discrepancy and its cause was understood, there was no need for extensive measurement campaigns to analyze every printed model.

A first model, displayed in Figure 4.22, showcases a unique 3D-printed design. The figure illustrates the successful completion of the printing process, with no issues arising from the complexity of the model. The printed piece is a seamless representation, devoid of any noticeable flaws.



(a) Front side

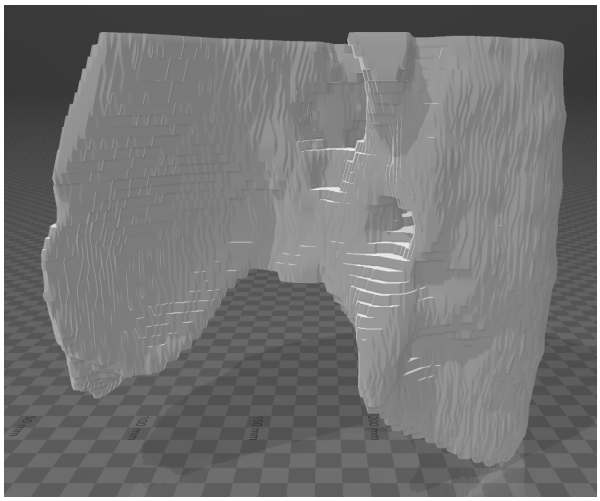
(b) Left side

(c) Back side

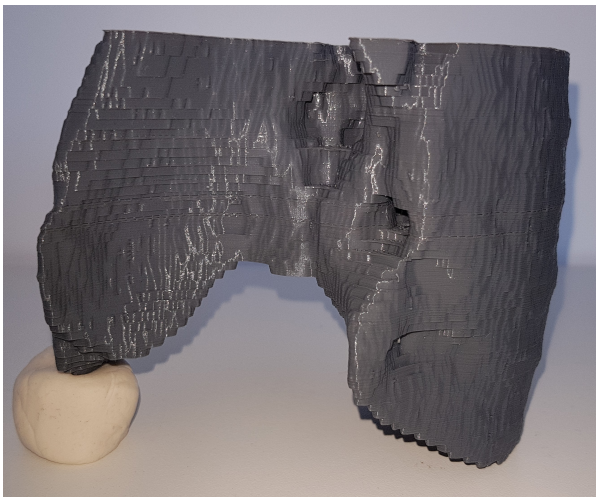
(d) Right side

Figure 4.22: 3D print of a model

For the followed examples, pictures of the printed models LUNG_L and LUNG_R will be displayed in Figures 4.23-4.26 alongside of their numerical rendering. We can observe that the textured appearance of models without any smoothing is accurately reproduced and does not pose any issues for the printing process. The same concept applies to the smoothed models. To facilitate the comparison, more photos of the printed models will be displayed alongside screenshots of the corresponding 3D models in a dedicated appendix section.

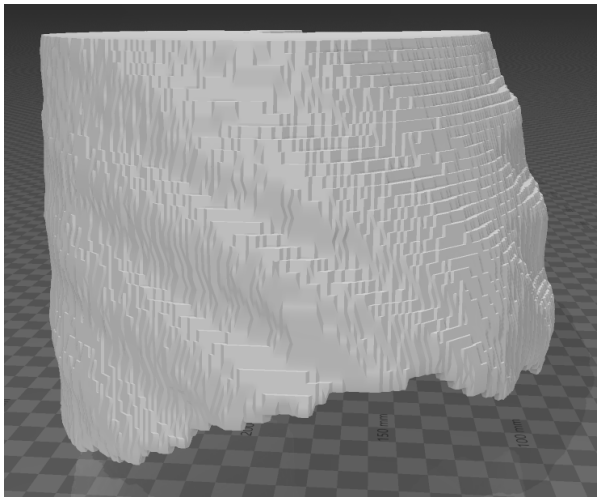


(a) Numerical Model : Front side

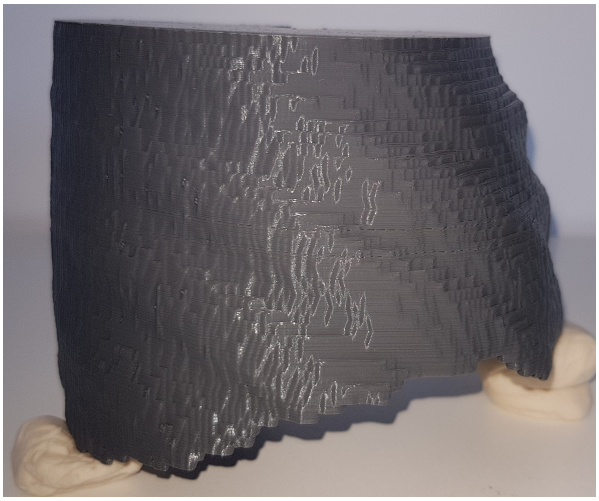


(b) Printed Model : Front side

Figure 4.23: Comparison of results of model LUNG_R built from DICOM RT Struct

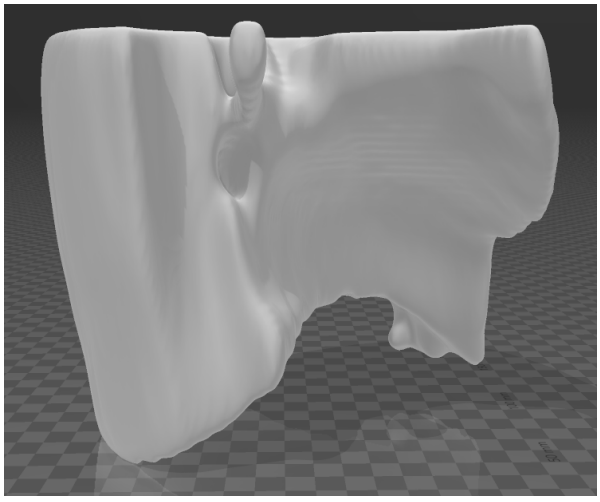


(a) Numerical Model : Back side

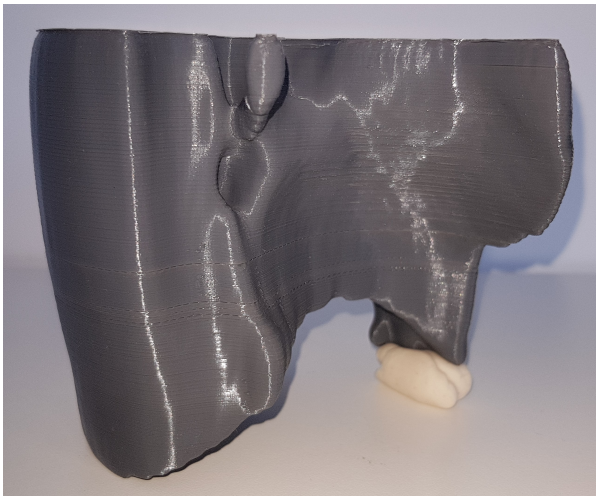


(b) Printed Model : Back side

Figure 4.24: Comparison of results of model LUNG_R built from DICOM RT Struct

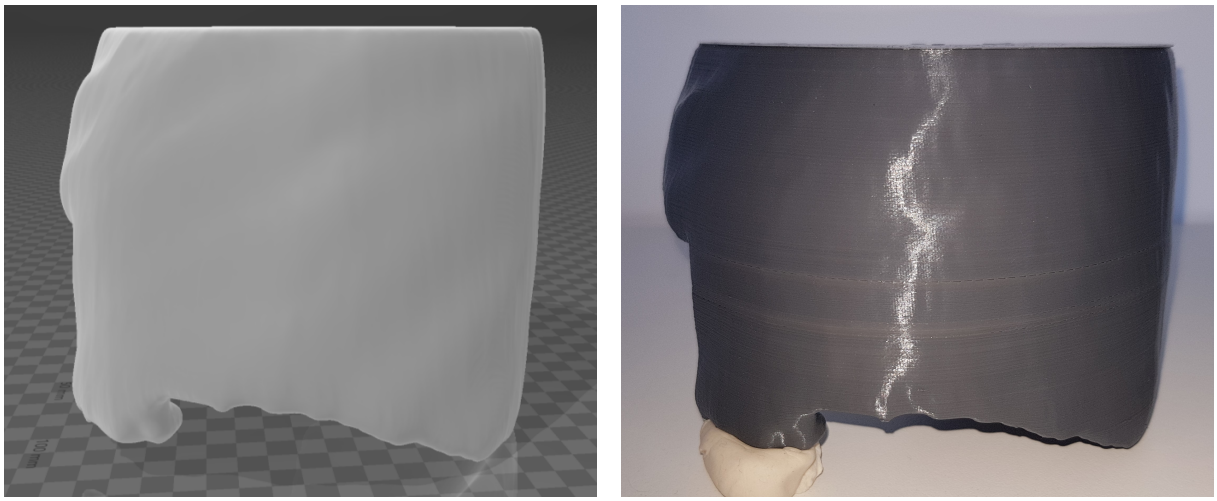


(a) Numerical Model : Front side



(b) Printed Model : Front side

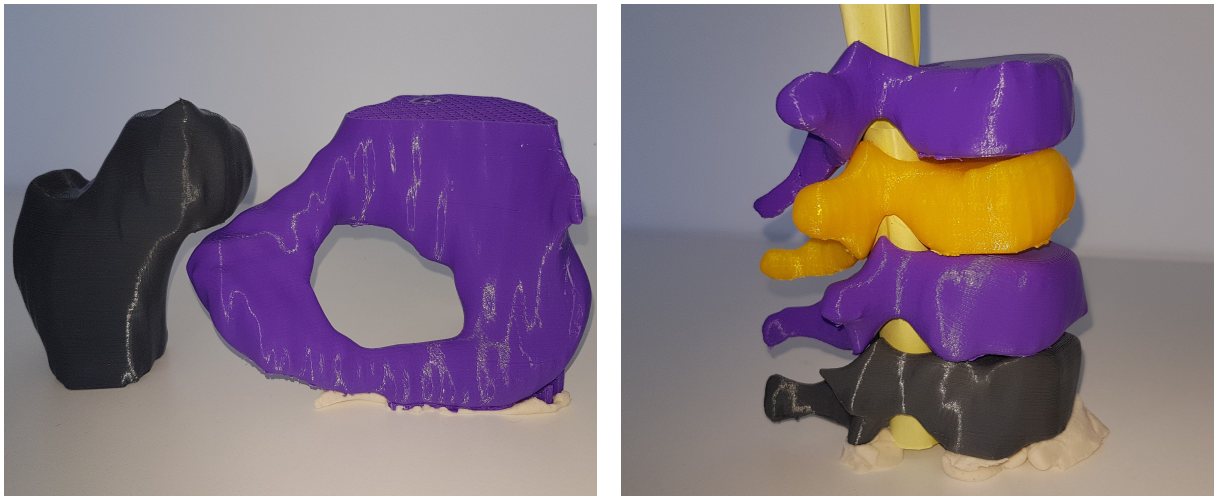
Figure 4.25: Comparison of results of model LUNG_L built from DICOM RT Struct



(a) Numerical Model : Back side (b) Printed Model : Back side

Figure 4.26: Comparison of results of model LUNG_L built from DICOM RT Struct

To illustrate the ability to combine models, additional examples of prints will be showcased. What makes these examples unique is that all the models displayed below were originally segmented by TotalSegmentator using CT scans. This section serves to demonstrate the successful execution of the processes described in the previous sections. On the left, we can observe the printed model of a femur head and the bottom part of a hip, while on the right, multiple vertebrae are printed to simulate a section of the spine.



(a) Models Name : Vertebrae T9-T12 (b) Models Name : Femur and Hip

Figure 4.27: 3D prints of models built from DICOM CT Scan serie

4.6.1 Problems

Of course, some problems could occur with 3D printers. All the problems that happen and that will be noticed in this thesis were always a problem from the printer itself. Two main classes of problems can be described.

The first type of problem occurs when the printers cease depositing plastic on the model currently being printed. This issue leads to a halt in the printing process, as the plastic being extruded from the printhead fails to adhere to anything. An example of such a problem is provided below. While it is not a serious concern, it does require restarting the print job, resulting in a loss of time. Constantly monitoring the print progress can help detect this problem earlier, but for prints that can take hours to complete, it would be impractical for a human to do so. The cause of this problem can sometimes be attributed to the

type of plastic being used, which may contain impurities, or it could be an error with the printer itself. One way to resolve this issue is by increasing the temperature of the printhead. This can potentially reduce errors, as the higher temperature makes the plastic more fluid, allowing it to adhere more easily to the model.

The second type of problem arises when the plastic fails to adhere properly to the printer's build plate. This can result in various issues, such as the model shifting during the printing process or parts of the support structure being unable to support future sections of the print. This problem is similar to the previous one, and sometimes users need to restart the print job. The root cause of this problem is similar to what was described earlier, and the solution remains the same. An example of such a problem is also illustrated in Figure 4.29.

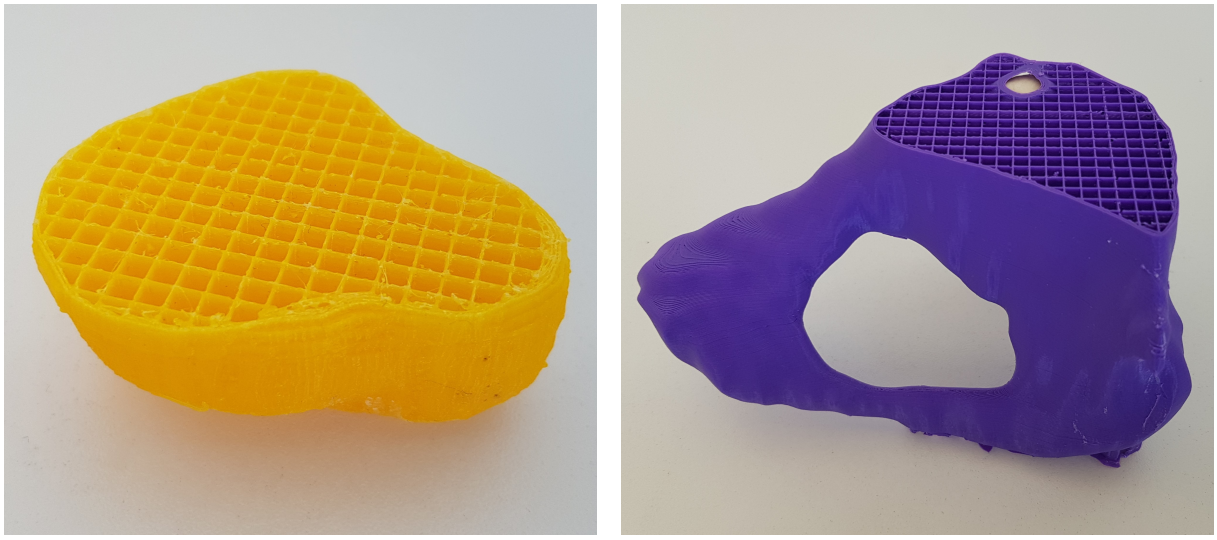


Figure 4.28: Examples for the first problem

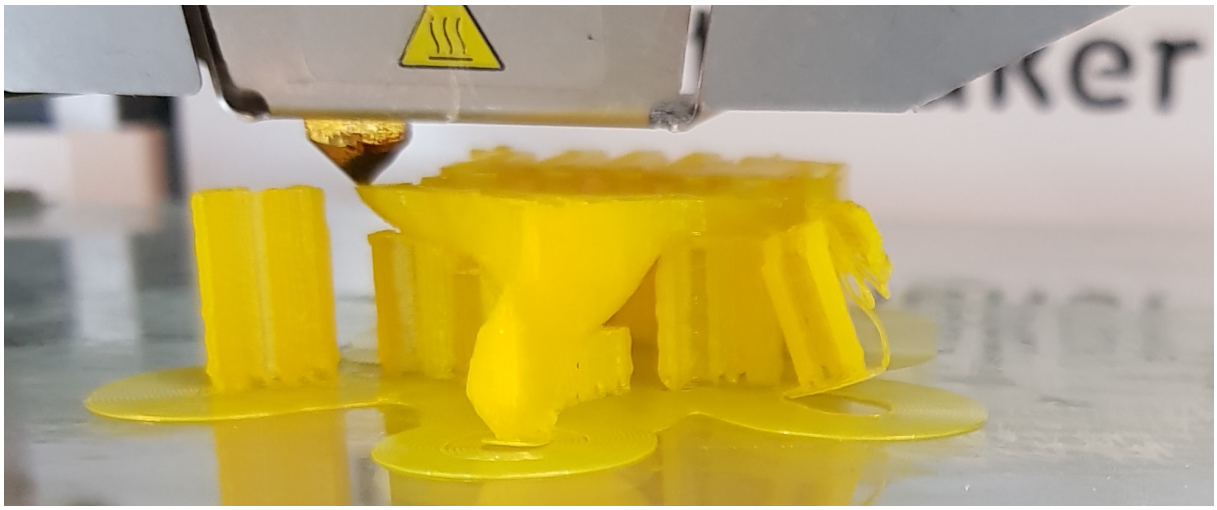


Figure 4.29: Example for the second problem

A last element which is not a problem but which has to be take into consideration, is that all the plastics used in 3D printers are not the same. Below are zoomed part of prints. The prints are the same piece but print with different materials. We can see that the gray one has more parts of lower quality than the yellow one.

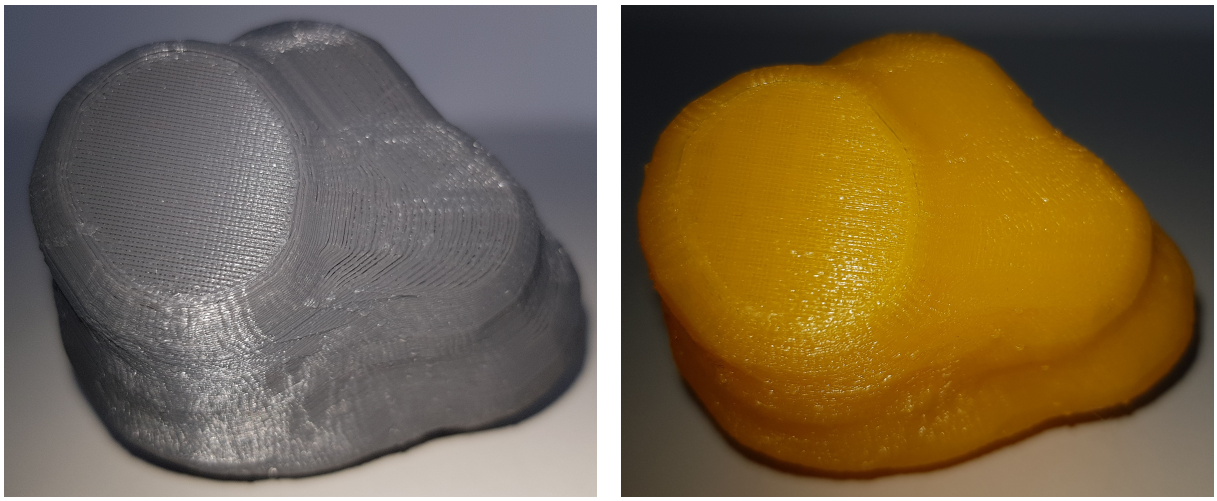


Figure 4.30: 3D prints of the same models with different kind of plastic

4.7 User Interface

The user interface created and all the codes are usable on Windows and Linux. This was done to make it easier to use for a large number of people. On the images in the Figures 4.31-4.35, screenshots of the interface are present. The first corresponds to a part that will always be visible on the interface. This is used to display information relating to the progress of the codes or to more general features of the interface. The other images present the different pages where the purpose of each page is understandable from the different buttons and configurable elements of the software.

This software and codes are Open Source and can be found with sample files to test functionality by following the GitHub link:

```
https://github.com/tgregoir-oss/DICOM_STL_Thesis
```

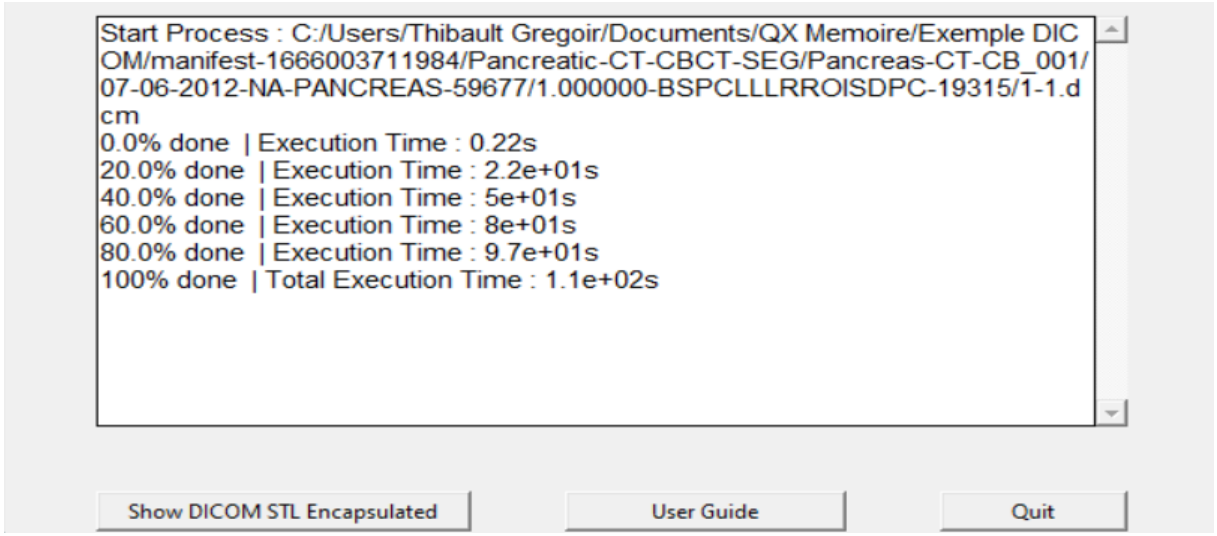


Figure 4.31: General elements present on each page with example of information

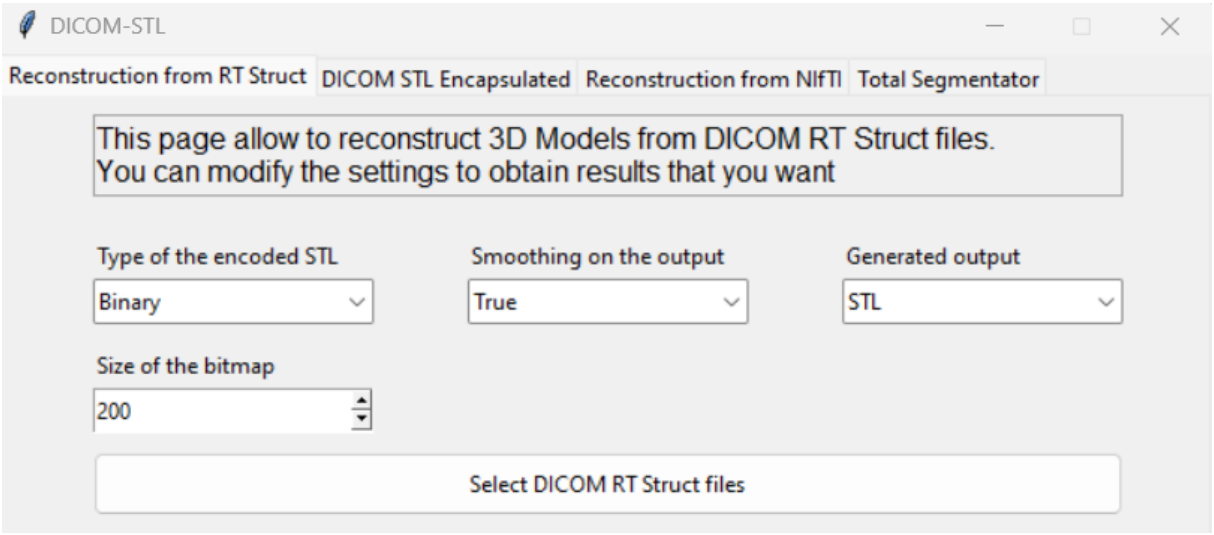


Figure 4.32: Page about reconstruction from DICOM RT Struct

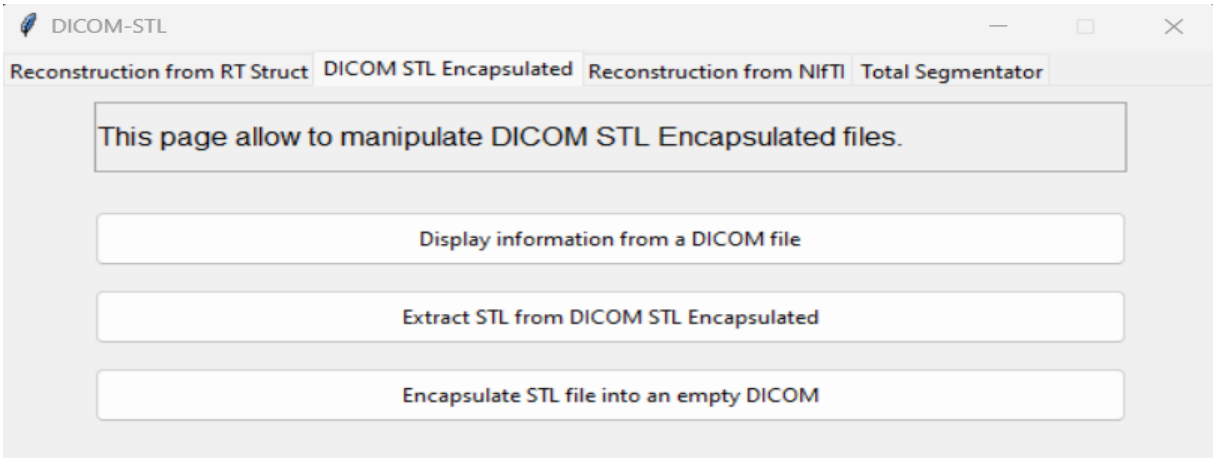


Figure 4.33: Page about manipulating DICOM

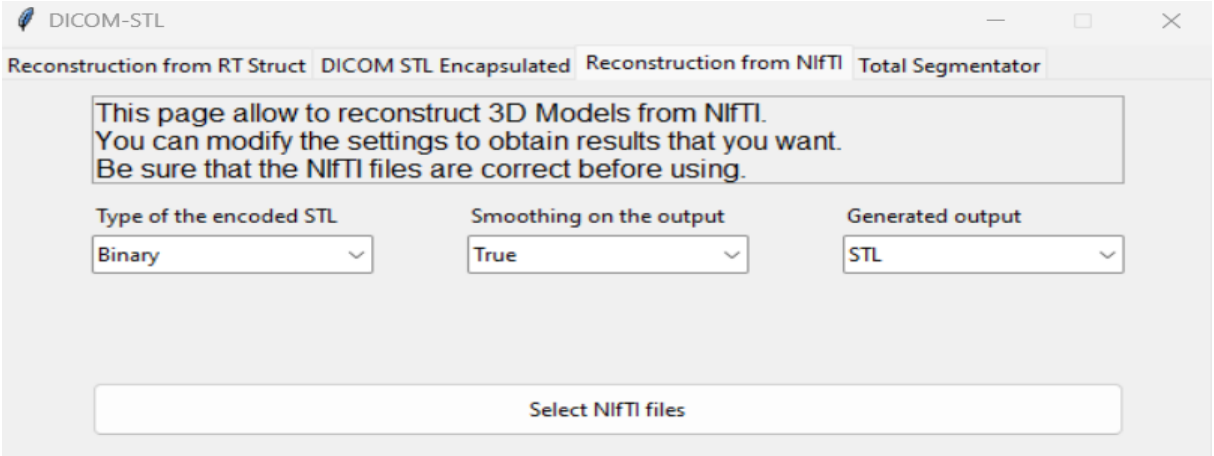


Figure 4.34: Page about reconstruction from NIfTI file

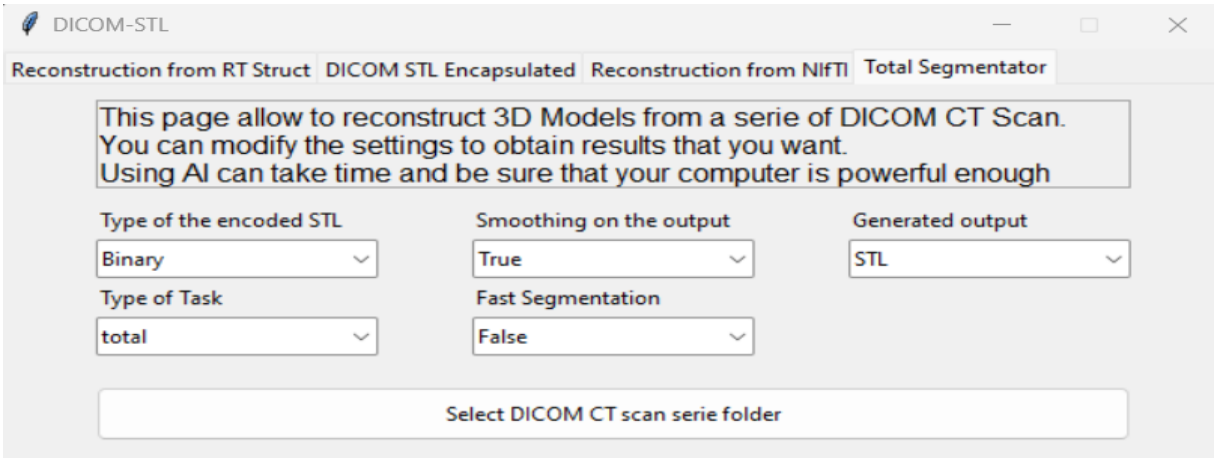


Figure 4.35: Page about TotalSegmentator

Conclusion

Contributions

The objective of this master's thesis was to validate the feasibility of printing and manipulating 3D models derived from manually or automatically segmented medical data. It was also needed to explore the interoperability of encoding the 3D models in the medical field. To achieve this, algorithm was developed and optimized to recreate 3D models from masks or contours. The implementation of various informatic tools and hardware was necessary to accomplish the different subtasks associated with this thesis.

Once the 3D models could be generated, the focus shifted towards exploring interoperability for handling, displaying, and encapsulating STL files. This involved creating DICOM STL Encapsulated files that underwent validation checks to ensure file completeness. Additionally, a web application was developed to enable data and 3D model visualization, providing a comprehensive user experience that leveraged these files standards.

The TotalSegmentator artificial intelligence (AI) was utilized to showcase the potential of AI in the medical field. Despite the considerable computational resources necessary for these tools, the capability to conduct 104 segmentations using AI and reconstruct those models using the code developed in the previous sections resulted in more efficient work compared to manual methods. This represents a significant improvement in terms of workload.

3D models were printed to demonstrate that physical objects can closely resemble to their virtual counterparts, thereby opening up possibilities for practical applications such as patient follow-up and treatment preparation. A set of printed models can then be assembled to build a faithful representation of a human body

Furthermore, a standalone and open-source software was developed to provide accessibility for anyone to utilize the findings of this thesis. The user interface was designed to be user-friendly and accessible to individuals of all backgrounds. This software can be easily installed and launched on both Windows and Linux operating systems, making it accessible to anyone with a computer.

Certainly, all these elements have their limitations, as will be discussed below. However, despite these challenges, the potential of these technologies is immense. In their current state, individuals such as doctors and technicians can utilize the software with just a few clicks and some processing time to obtain a multitude of files and models. These outputs can be utilized in various medical applications or easily integrated into other tools for further analysis and utilization. In conclusion, the objective of this thesis has been reached.

Discussion

This section will discuss about certain limitations pertaining to the choices and technologies utilized in this master's thesis. The intention is not to undermine the entire work but rather to provide an explanation for certain aspects. Although the results are promising, it is important to address certain drawbacks that place the work within a realistic context.

A general observation is that the set of tasks performed is more like a "proof of concept" than a fully-fledged application that one might find online. While all the objectives have been achieved, it could be considered limited in view of the capabilities offered by the code and the software.

Ethical aspect

By providing easy access to powerful tools like TotalSegmentator, we not only enable widespread use but also open the door to potential advancements in the near future. As described, artificial intelligence can be employed to reconstruct the skin or organs of individuals who have undergone a CT scan. Moreover, if this data were to be leaked, it could be easier to re-identify individuals and learn specific details about their bodies. This could lead to serious privacy concerns. For example, malicious individuals could potentially use this information to blackmail patients or sell the data of famous individuals on the internet if they gain access to this technology.

The issue is similar for DICOM RT Struct files, even though they typically contain less data about 3D models compared to what TotalSegmentator can generate. However, the problem of medical data leakage is a significant concern that should not be underestimated.

Choice of the Algorithm

For this master's thesis, we have selected the Marching Cubes algorithm. The rationale behind this choice was explained in the methodology section. However, it is worth noting that alternative algorithms may yield better results within the same time frame. The input data was obtained in the form of contours since it is how the data is displayed in the RT Struct files. Considering the data as a point cloud rather than contours could potentially yield different results. Also, this algorithm may take some time to run. Other methods may provide similar or better results in just a few seconds.

The Marching Cubes algorithm is effective in reconstructing 3D surfaces from points, making it suitable for visualization purposes. However, if the objective is to design implants or to precisely locate tumors, an alternative method should be employed. While the 3D models generated by the Marching Cubes algorithm capture the overall shape, the vertices of the triangles may not perfectly align with the points obtained from the RT Struct file. Although the results may be more realistic when using the data provided by TotalSegmentator, some degree of inaccuracy may still be present. It is also important to recognize that errors resulting from AI can still occur.

Assumptions on the Data

Throughout all the tests conducted, the contours stored in the RT Struct files exhibited a consistent organizational structure. Each subsequent point allowed for the correct rendering of the contours, with the last point in an array also serving as the first point. This feature facilitated the easy drawing of contours on a bitmap. No function was developed to sort the points if they were originally positioned randomly within the file.

Additionally, an assumption was made that each point on a contour would share a consistent value on a same axis. However, no verification was performed to determine on which axis the shared value should be or if all points indeed shared the same value. Although no issues were encountered during testing, it is important to acknowledge that these are considered edge cases. Although these cases can happen, nothing has been done to handle these situations because it was assumed that the creators of the files had to follow certain rules, even implicit ones. Methods to resolve these assumptions could be implemented in the code without rewriting all the algorithms

Choice of some tools for this master thesis

For the web application, we utilized Axios and Three.js in JavaScript, Flask and Tkinter in Python for creating the user interface and the codes of the tool. These choices were based on the availability of tutorials and good documentation for these tools. They also met our requirements without the need for extensive additional components such as downloads or configurations. However, no comparison was conducted between similar technologies to determine performance or ease of use.

Choice of the AI

The selection of the AI model was primarily based on it being a free tool, easily usable via the Python API, and seamlessly integrable into the existing codebase due to the output it produced. Utilizing

alternative U-net models or developing a new one could be potential avenues for future research and expansion.

Future Works

There are still unexplored possibilities, and this work could serve as a foundation for future contributions in the field of medical imaging. Here are some potential areas of interest for future research and development:

- Is the Marching Cubes algorithm the optimal choice for recreating anatomical structures, or could other algorithms potentially yield better results?
- What techniques, such as multithreading or preprocessing methods, could be employed to enhance the performance and outcomes of the Marching Cubes algorithm?
- Given that the 3D printings were conducted solely on Ultimaker printers, what influence might different printer models and additional printable resources have on the resulting prints?
- What could be the tangible impact of 3D prints and 3D models in aiding patients during medical treatments?
- The current web application used for displaying 3D models lacks certain features, such as the ability to highlight models or draw on the scene. How can a more comprehensive, optimized, and open-source web application be developed to effectively display STL files and provide enhanced functionality?
- The artificial intelligence employed in this thesis may require substantial computational resources. What approaches can be implemented to improve the performance of TotalSegmentator and optimize its computational requirements?
- Could another artificial intelligence model, whether built from scratch or utilizing existing models, potentially outperform TotalSegmentator?
- The outcome of this thesis is a standalone open-source software written in Python. Could a similar project written in another programming language offer improved performance? Additionally, could such a project be integrated into well-known open-source software platforms like Orthanc?

This master's thesis serves as a valuable proof of concept for various aspects of 3D reconstructions from medical images. However, there are still numerous areas to explore and improve upon. The field of AI is continuously evolving, and these technologies hold immense potential across a wide range of domains in the future.

Bibliography

[1] Tran Kai Frank Da and David Cohen-Steiner. "Advancing Front Surface Reconstruction". In: *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgAdvancingFrontSurfaceReconstruction.

[2] Liangliang Nan. "Polygonal Surface Reconstruction". In: *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgPolygonalSurfaceReconstruction.

[3] Yukie Nagai, Yutaka Ohtake, and Hiromasa Suzuki. "Tomographic surface reconstruction from point cloud". In: *Computers & Graphics*. Shape Modeling International 2014 46 (Feb. 1, 2015), pp. 55–63. ISSN: 0097-8493. DOI: 10.1016/j.cag.2014.09.034. URL: https://www.sciencedirect.com/science/article/pii/S0097849314001228 (visited on 05/23/2023).

[4] Priyanka Mandikal et al. *3D-LMNet: Latent Embedding Matching for Accurate and Diverse 3D Point Cloud Reconstruction from a Single Image*. Mar. 26, 2019. DOI: 10.48550/arXiv.1807.07796. arXiv: 1807.07796[cs]. URL: http://arxiv.org/abs/1807.07796 (visited on 04/29/2023).

[5] Francis Engelmann et al. *From Points to Multi-Object 3D Reconstruction*. June 21, 2021. DOI: 10.48550/arXiv.2012.11575. arXiv: 2012.11575[cs]. URL: http://arxiv.org/abs/2012.11575 (visited on 04/29/2023).

[6] Kam-sum Lee et al. "3D Face Modeling from Perspective-Views and Contour-Based Generic-Model". In: *Real-Time Imaging* 7.2 (Apr. 1, 2001), pp. 173–182. ISSN: 1077-2014. DOI: 10.1006/rtim.2000.0241. URL: https://www.sciencedirect.com/science/article/pii/S1077201400902419 (visited on 05/03/2023).

[7] Mona Eulitz and Gebhard Reiss. "3D reconstruction of SEM images by use of optical photogrammetry software". In: *Journal of Structural Biology* 191.2 (Aug. 1, 2015), pp. 190–196. ISSN: 1047-8477. DOI: 10.1016/j.jsb.2015.06.010. URL: https://www.sciencedirect.com/science/article/pii/S1047847715300149 (visited on 04/29/2023).

[8] Po-Hao Huang and Shang-Hon Lai. "Contour-Based Structure from Reflection". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). Vol. 1. ISSN: 1063-6919. June 2006, pp. 379–386. DOI: 10.1109/CVPR.2006.88.

[9] Benjamin Ummenhofer and Thomas Brox. "Point-Based 3D Reconstruction of Thin Objects". In: *2013 IEEE International Conference on Computer Vision*. 2013 IEEE International Conference on Computer Vision. ISSN: 2380-7504. Dec. 2013, pp. 969–976. DOI: 10.1109/ICCV.2013.124.

[10] Mesapam Shashi and Kamal Jain. "Use of photogrammetry in 3D modeling and visualization of buildings". In: 2 (Jan. 1, 2007).

[11] Priyanka Mandikal and R. Venkatesh Babu. *Dense 3D Point Cloud Reconstruction Using a Deep Pyramid Network*. Jan. 25, 2019. DOI: 10.48550/arXiv.1901.08906. arXiv: 1901.08906[cs]. URL: http://arxiv.org/abs/1901.08906 (visited on 05/23/2023).

[12] F. Remondino et al. "UAV PHOTOGRAMMETRY FOR MAPPING AND 3D MODELING – CURRENT STATUS AND FUTURE PERSPECTIVES". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XXXVIII-1-C22 (Sept. 6, 2012). Conference Name: International Conference on Unmanned Aerial Vehicle in Geomatics (UAV-g) (Volume XXXVIII-1/C22) - 14–16 September, Zurich, Switzerland Publisher: Copernicus GmbH, pp. 25–31. ISSN: 1682-1750. DOI: 10.5194/isprsarchives-XXXVIII-1-C22-25-2011. URL: https://isprs-archives.copernicus.org/articles/XXXVIII-1-C22/25/2011/ (visited on 04/29/2023).

[13] Matthew Berger et al. “A Survey of Surface Reconstruction from Point Clouds”. In: *Computer Graphics Forum* 36.1 (2017). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12802, pp. 301–329. ISSN: 1467-8659. DOI: 10.1111/cgf.12802. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12802 (visited on 05/02/2023).

[14] Cheng Chun You et al. “A Survey on Surface Reconstruction Techniques for Structured and Unstructured Data”. In: *2020 IEEE Conference on Open Systems (ICOS)*. 2020 IEEE Conference on Open Systems (ICOS). ISSN: 2473-3660. Nov. 2020, pp. 37–42. DOI: 10.1109/ICOS50156.2020.9293685.

[15] Amin Alqudah. “Survey of Surface Reconstruction Algorithms”. In: *Journal of Signal and Information Processing* 5.3 (July 30, 2014). Number: 3 Publisher: Scientific Research Publishing, pp. 63–79. DOI: 10.4236/jsip.2014.53009. URL: http://www.scirp.org/Journal/Paperabs.aspx?paperid=48300 (visited on 05/02/2023).

[16] Alireza Khatamian and Hamid R. Arabnia. “Survey on 3D Surface Reconstruction”. In: *Journal of Information Processing Systems* 12.3 (Sept. 1, 2016), pp. 338–357. URL: http://jips-k.org/q.jips?cp=pp&pn=399 (visited on 05/02/2023).

[17] Baoyun Guo et al. “A 3D Surface Reconstruction Method for Large-Scale Point Cloud Data”. In: *Mathematical Problems in Engineering* 2020 (Aug. 6, 2020). Publisher: Hindawi, e8670151. ISSN: 1024-123X. DOI: 10.1155/2020/8670151. URL: https://www.hindawi.com/journals/mpe/2020/8670151/ (visited on 05/23/2023).

[18] Baorui Ma et al. *Surface Reconstruction from Point Clouds by Learning Predictive Context Priors*. Apr. 23, 2022. DOI: 10.48550/arXiv.2204.11015. arXiv: 2204.11015[cs]. URL: http://arxiv.org/abs/2204.11015 (visited on 05/23/2023).

[19] Martin A. Fischler and Robert C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (June 1, 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://dl.acm.org/doi/10.1145/358669.358692 (visited on 05/23/2023).

[20] Yuchen He et al. “Fast Algorithms for Surface Reconstruction from Point Cloud”. In: *Mathematical Methods in Image Processing and Inverse Problems*. Ed. by Xue-Cheng Tai, Suhua Wei, and Haiguang Liu. Springer Proceedings in Mathematics & Statistics. Singapore: Springer, 2021, pp. 61–80. ISBN: 9789811627019. DOI: 10.1007/978-981-16-2701-9_4.

[21] G. Saravana Kumar, Prem Kumar Kalra, and Sanjay G. Dhande. “Curve and surface reconstruction from points: an approach based on self-organizing maps”. In: *Applied Soft Computing* 5.1 (Dec. 1, 2004), pp. 55–66. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2004.04.003. URL: https://www.sciencedirect.com/science/article/pii/S1568494604000572 (visited on 05/23/2023).

[22] Bigui Lin, Dadi Jin, and Miguel Angel Socorro Borges. “3D surface reconstruction of the femur and tibia from parallel 2D contours”. In: *Journal of Orthopaedic Surgery and Research* 17.1 (Mar. 5, 2022), p. 145. ISSN: 1749-799X. DOI: 10.1186/s13018-022-02994-w. URL: https://doi.org/10.1186/s13018-022-02994-w (visited on 05/03/2023).

[23] David Meyers, Shelley Skinner, and Kenneth Sloan. “Surfaces from contours”. In: *ACM Transactions on Graphics* 11.3 (July 1, 1992), pp. 228–258. ISSN: 0730-0301. DOI: 10.1145/130881.131213. URL: https://dl.acm.org/doi/10.1145/130881.131213 (visited on 05/03/2023).

[24] H. Fuchs, Z. M. Kedem, and S. P. Uselton. “Optimal surface reconstruction from planar contours”. In: *Communications of the ACM* 20.10 (Oct. 1, 1977), pp. 693–702. ISSN: 0001-0782. DOI: 10.1145/359842.359846. URL: https://dl.acm.org/doi/10.1145/359842.359846 (visited on 05/03/2023).

[25] Byeong-Seok Shin and Hoe Sang Jung. “Fast Reconstruction of 3D Terrain Model from Contour Lines on 2D Maps”. In: *Systems Modeling and Simulation: Theory and Applications*. Ed. by Doo-Kwon Baik. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 230–239. ISBN: 978-3-540-30585-9. DOI: 10.1007/978-3-540-30585-9_26.

[26] H. N. Christiansen and T. W. Sederberg. “Conversion of complex contour line definitions into polygonal element mosaics”. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’78. New York, NY, USA: Association for Computing Machinery, Aug. 23, 1978, pp. 187–192. ISBN: 978-1-4503-7908-3. DOI: 10.1145/800248.807388. URL: https://dl.acm.org/doi/10.1145/800248.807388 (visited on 05/03/2023).

[27] James D. Fix and Richard E. Ladner. "Multiresolution banded refinement to accelerate surface reconstruction from polygons". In: *Computational Geometry* 13.1 (May 1, 1999), pp. 49–64. ISSN: 0925-7721. DOI: 10.1016/S0925-7721(99)00008-5. URL: https://www.sciencedirect.com/science/article/pii/S0925772199000085 (visited on 05/03/2023).

[28] G. Cong and B. Parvin. "Robust and efficient surface reconstruction from contours". In: *The Visual Computer* 17.4 (June 1, 2001), pp. 199–208. ISSN: 1432-2315. DOI: 10.1007/PL00013409. URL: https://doi.org/10.1007/PL00013409 (visited on 05/03/2023).

[29] O. Nilsson, D. Breen, and K. Museth. "Surface reconstruction via contour metamorphosis: an Eulerian approach with Lagrangian particle tracking". In: *VIS 05. IEEE Visualization, 2005.* VIS 05. IEEE Visualization, 2005. Oct. 2005, pp. 407–414. DOI: 10.1109/VISUAL.2005.1532823.

[30] J. Jeong et al. "A New Method for Solving Branching Problems in Surface Reconstruction". In: *The International Journal of Advanced Manufacturing Technology* 16.4 (Mar. 1, 2000), pp. 259–264. ISSN: 1433-3015. DOI: 10.1007/s001700050154. URL: https://doi.org/10.1007/s001700050154 (visited on 05/03/2023).

[31] Kailash Jha. "Construction of Branching Surface from 2-D Contours". In: ().

[32] Kailash Jha. "Reconstruction of Branching Surface and Its Smoothness by Reversible Catmull-Clark Subdivision". In: *Computational Science – ICCS 2009*. Ed. by Gabrielle Allen et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 759–769. ISBN: 978-3-642-01973-9. DOI: 10.1007/978-3-642-01973-9_85.

[33] Meihe Xu and Zesheng Tang. "Surface reconstruction for cross sectional data". In: *Journal of Computer Science and Technology* 11.5 (Sept. 1, 1996), pp. 471–479. ISSN: 1860-4749. DOI: 10.1007/BF02947214. URL: https://doi.org/10.1007/BF02947214 (visited on 05/03/2023).

[34] Desheng Wang et al. "Efficient surface reconstruction from contours based on two-dimensional Delaunay triangulation". In: *International Journal for Numerical Methods in Engineering* 65.5 (2006). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1465, pp. 734–751. ISSN: 1097-0207. DOI: 10.1002/nme.1465. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1465 (visited on 05/03/2023).

[35] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction". In: *Symposium on Geometry Processing*. Ed. by Alla Sheffer and Konrad Polthier. The Eurographics Association, 2006. ISBN: 3-905673-24-X. DOI: 10.2312/SGP/SGP06/061-070.

[36] F. Bernardini et al. "The ball-pivoting algorithm for surface reconstruction". In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (Oct. 1999). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 349–359. ISSN: 1941-0506. DOI: 10.1109/2945.817351.

[37] Yi An et al. "Self-adaptive polygon mesh reconstruction based on ball-pivoting algorithm". In: *International Journal of Computer Applications in Technology* 54.1 (Jan. 2016). Publisher: Inderscience Publishers, pp. 51–60. ISSN: 0952-8091. DOI: 10.1504/IJCAT.2016.077790. URL: https://www.inderscienceonline.com/doi/abs/10.1504/IJCAT.2016.077790 (visited on 05/01/2023).

[38] A.R. Specht and M. Devy. "Surface segmentation using a modified ball-pivoting algorithm". In: *2004 International Conference on Image Processing, 2004. ICIP '04.* 2004 International Conference on Image Processing, 2004. ICIP '04. Vol. 3. ISSN: 1522-4880. Oct. 2004, 1931–1934 Vol. 3. DOI: 10.1109/ICIP.2004.1421457.

[39] Juan Fen Jiang, Yue Qi Zhong, and Qiu Ping Zhang. "Three-Dimensional Garment Surface Reconstruction Based on Ball-Pivoting Algorithm". In: *Advanced Materials Research* 821-822 (2013). Publisher: Trans Tech Publications Ltd, pp. 765–768. ISSN: 1662-8985. DOI: 10.4028/www.scientific.net/AMR.821-822.765. URL: https://www.scientific.net/AMR.821-822.765 (visited on 05/01/2023).

[40] Wei Ma and Qingquan Li. "An Improved Ball Pivot Algorithm-Based Ground Filtering Mechanism for LiDAR Data". In: *Remote Sensing* 11.10 (Jan. 2019). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 1179. ISSN: 2072-4292. DOI: 10.3390/rs11101179. URL: https://www.mdpi.com/2072-4292/11/10/1179 (visited on 05/01/2023).

[41] Julie Digne. "An Analysis and Implementation of a Parallel Ball Pivoting Algorithm". In: *Image Processing On Line* 4 (July 1, 2014), pp. 149–168. ISSN: 2105-1232. DOI: 10.5201/ipol.2014.81. URL: https://www.ipol.im/pub/art/2014/81 (visited on 05/01/2023).

[42] William E. Lorensen and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM SIGGRAPH Computer Graphics* 21.4 (Aug. 1, 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422. URL: https://dl.acm.org/doi/10.1145/37402.37422 (visited on 04/17/2023).

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[44] Fabian Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". In: *Nature Methods* 18.2 (Feb. 2021). Number: 2 Publisher: Nature Publishing Group, pp. 203–211. ISSN: 1548-7105. DOI: 10.1038/s41592-020-01008-z. URL: https://www.nature.com/articles/s41592-020-01008-z (visited on 04/17/2023).

[45] Jakob Wasserthal et al. *TotalSegmentator: robust segmentation of 104 anatomical structures in CT images*. Aug. 11, 2022. DOI: 10.48550/arXiv.2208.05868. arXiv: 2208.05868[cs,eess]. URL: http://arxiv.org/abs/2208.05868 (visited on 06/04/2023).

Appendix A

Comparison between the original models and the smoothed version

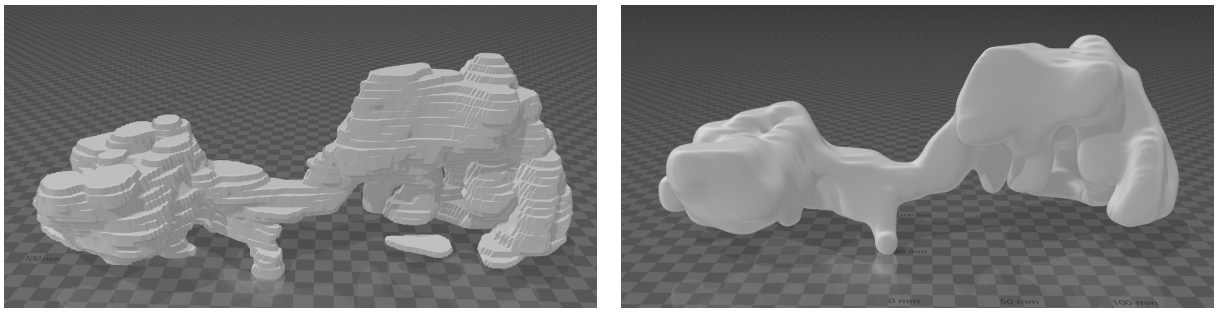


Figure A.1: Model name : Bowel_sm_planCT

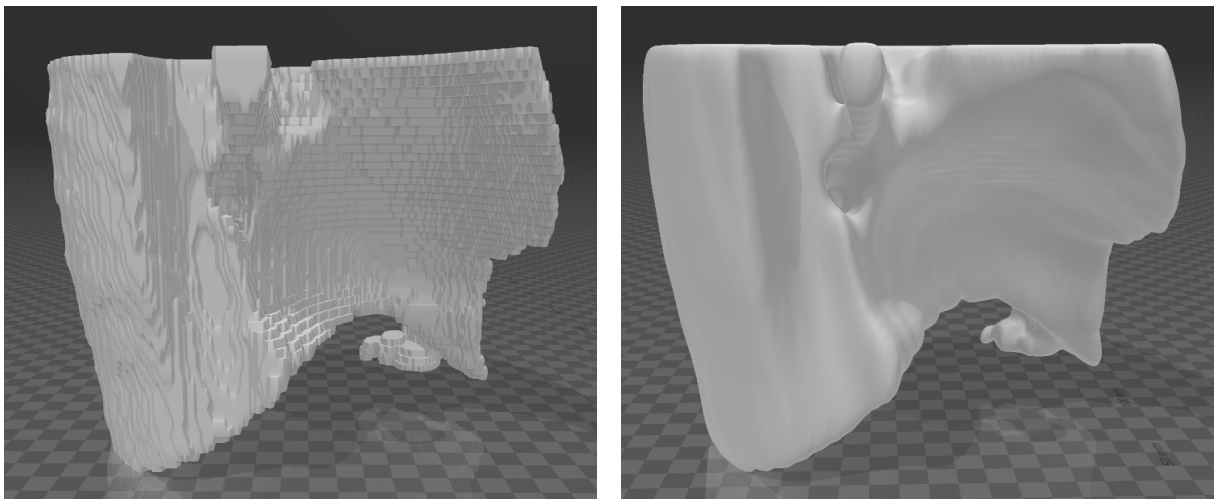


Figure A.2: Model name : LUNG_L

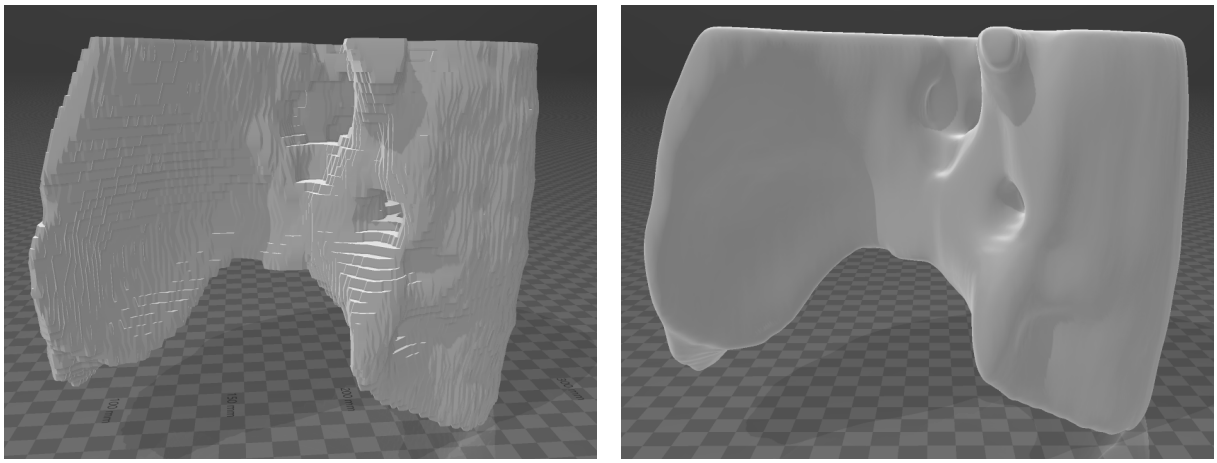


Figure A.3: Model name : LUNG_R

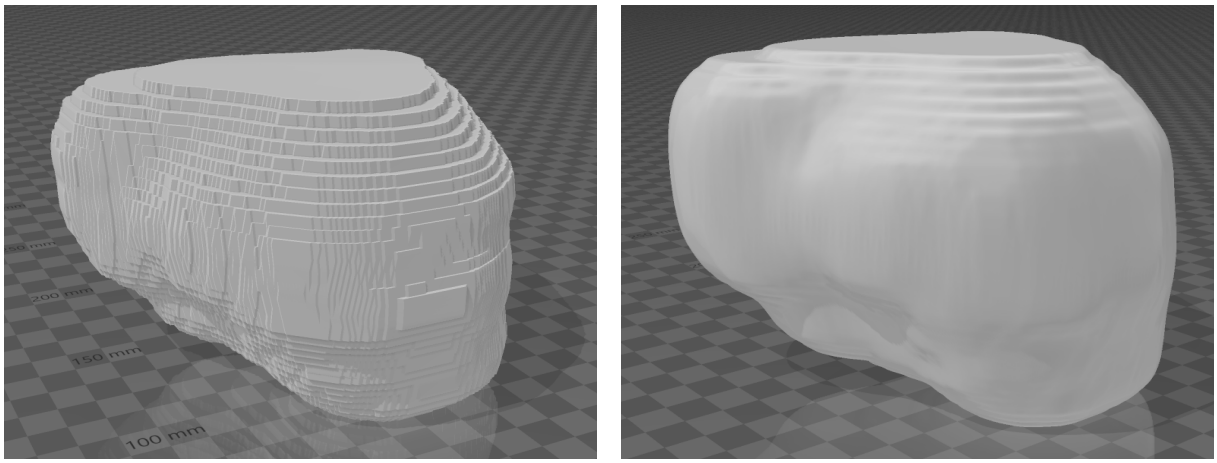


Figure A.4: Model name : ROI

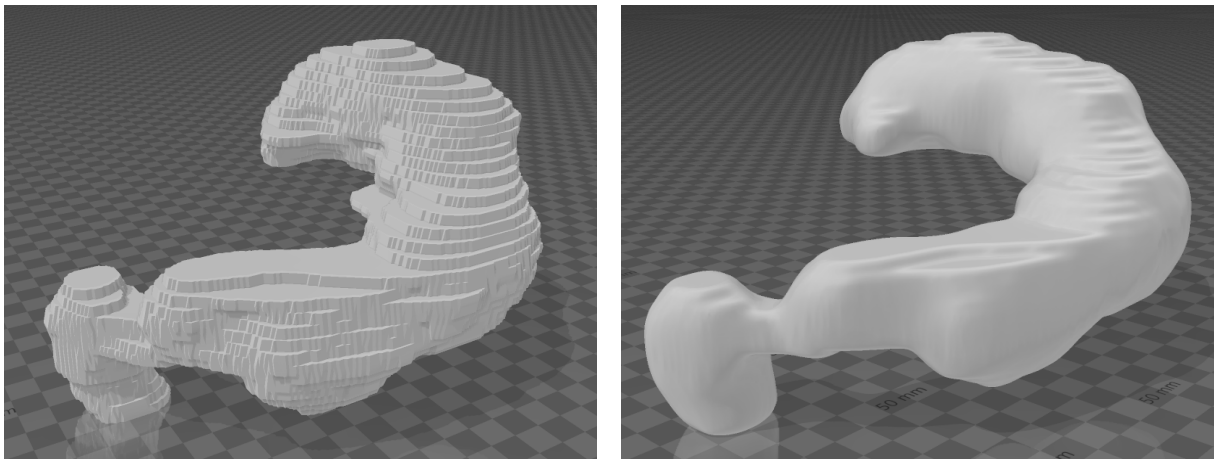
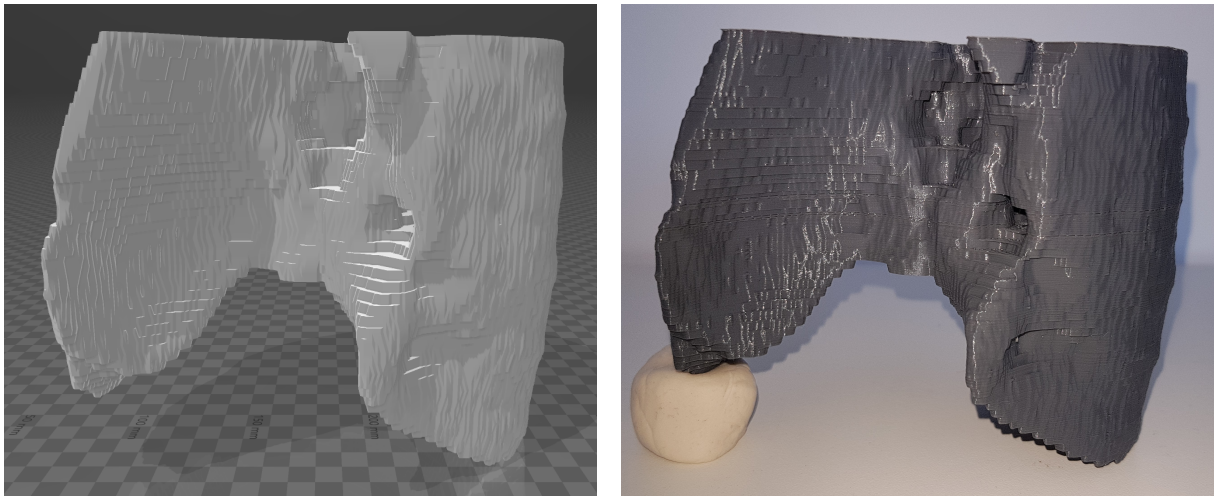


Figure A.5: Model name : Stomach_duo_planCT

Appendix B

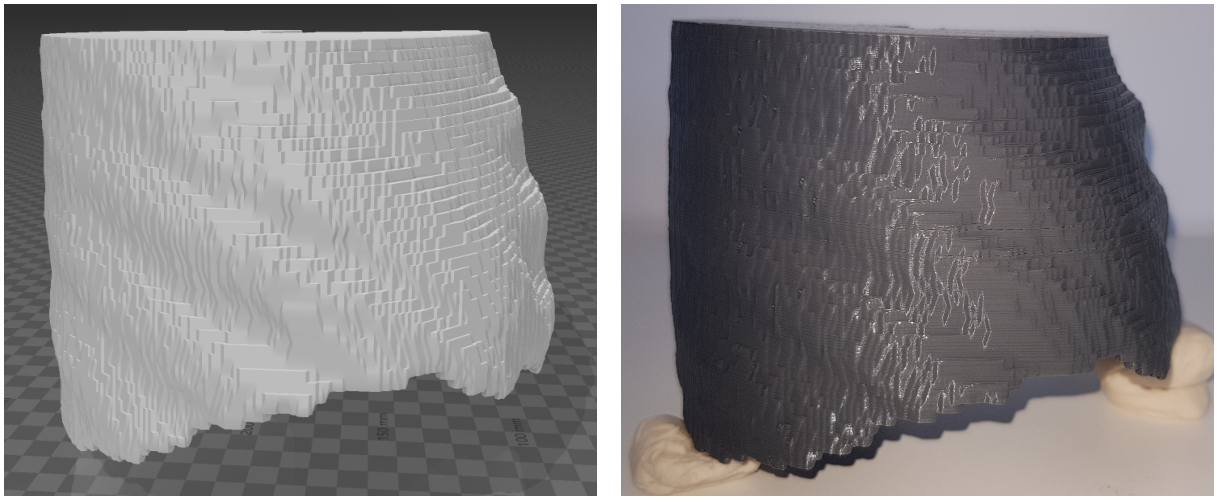
Comparison between the printed models and their corresponding digital version

B.1 LUNG_R Model



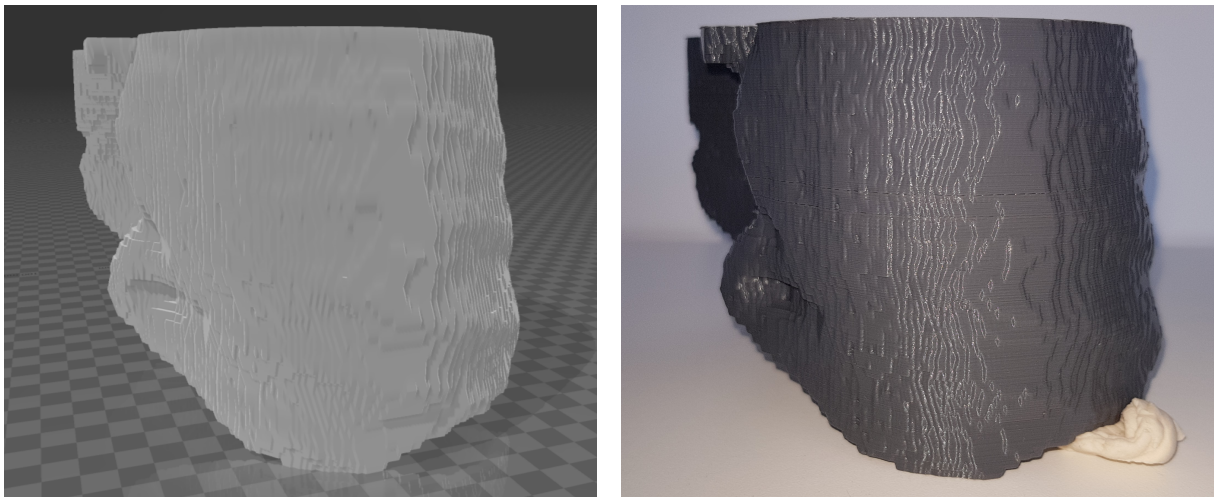
(a) Numerical Model (b) Printed Model

Figure B.1: Comparison of the front side



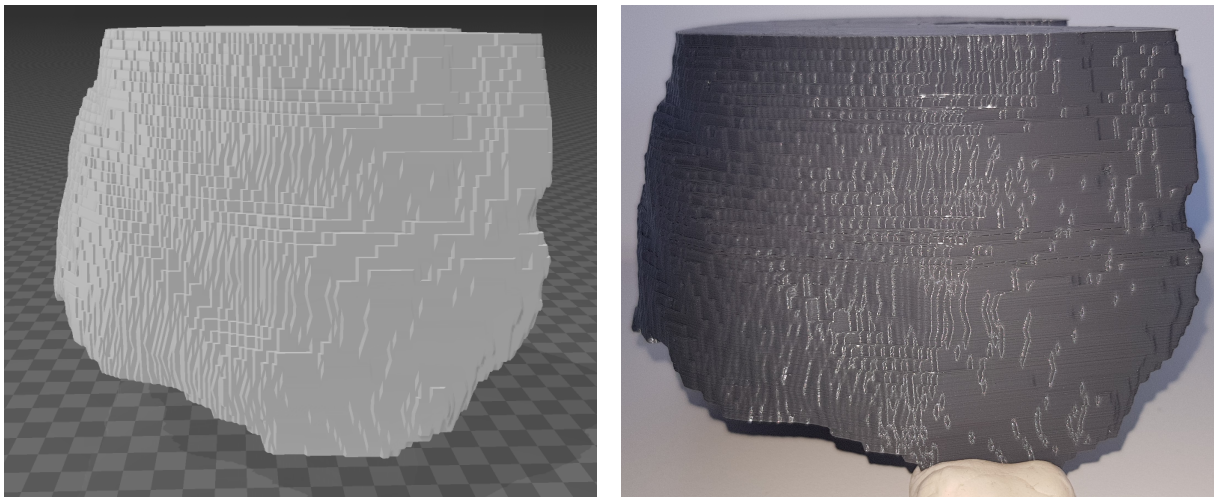
(a) Numerical Model (b) Printed Model

Figure B.2: Comparison of the back side



(a) Numerical Model (b) Printed Model

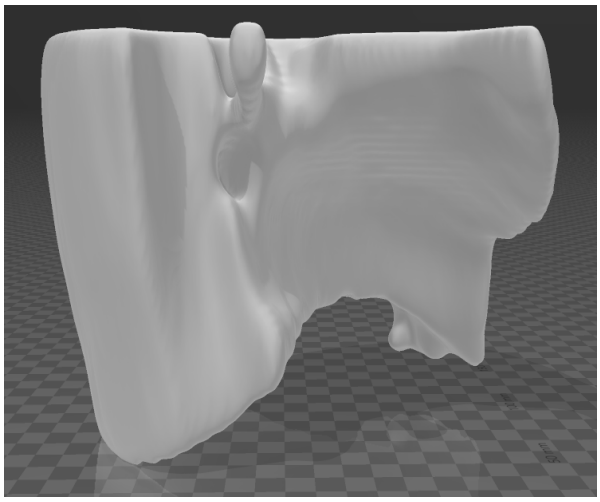
Figure B.3: Comparison of the left side



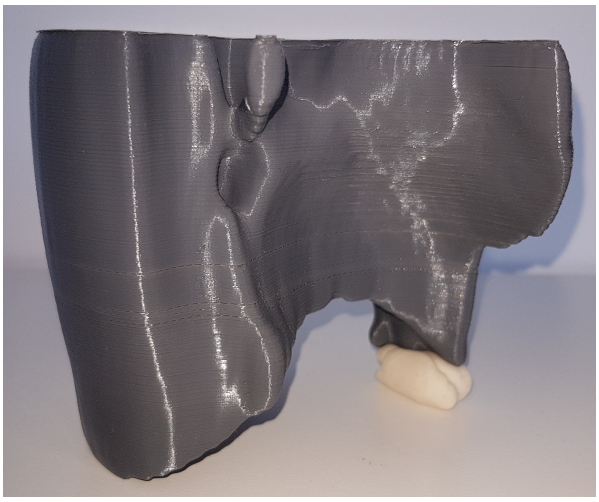
(a) Numerical Model (b) Printed Model

Figure B.4: Comparison of the right side

B.2 LUNG_L Model

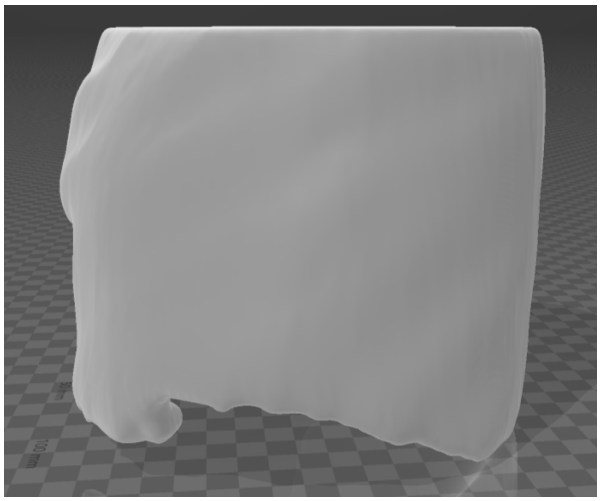


(a) Numerical Model

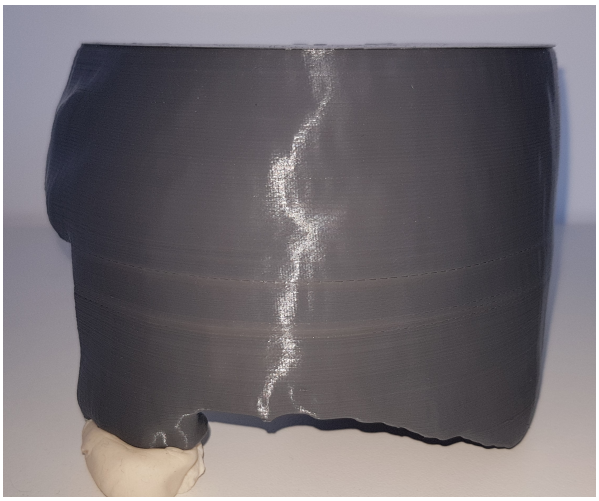


(b) Printed Model

Figure B.5: Comparison of the front side

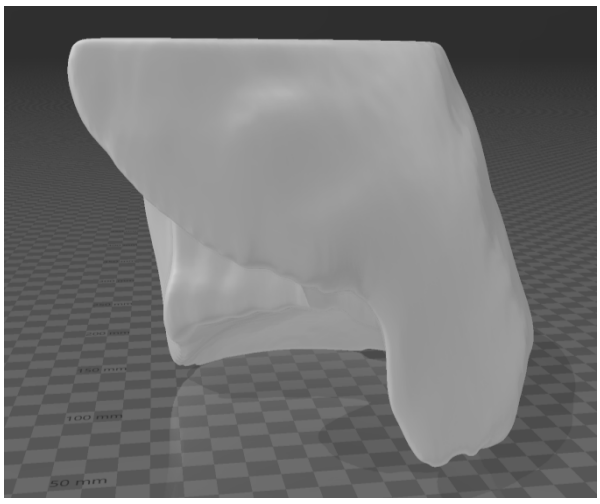


(a) Numerical Model

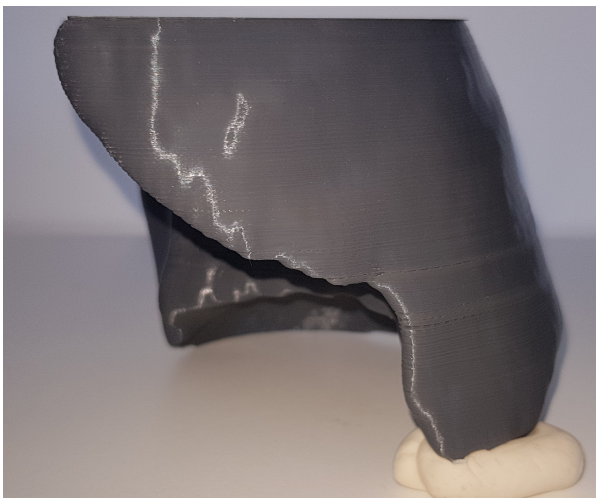


(b) Printed Model

Figure B.6: Comparison of the back side

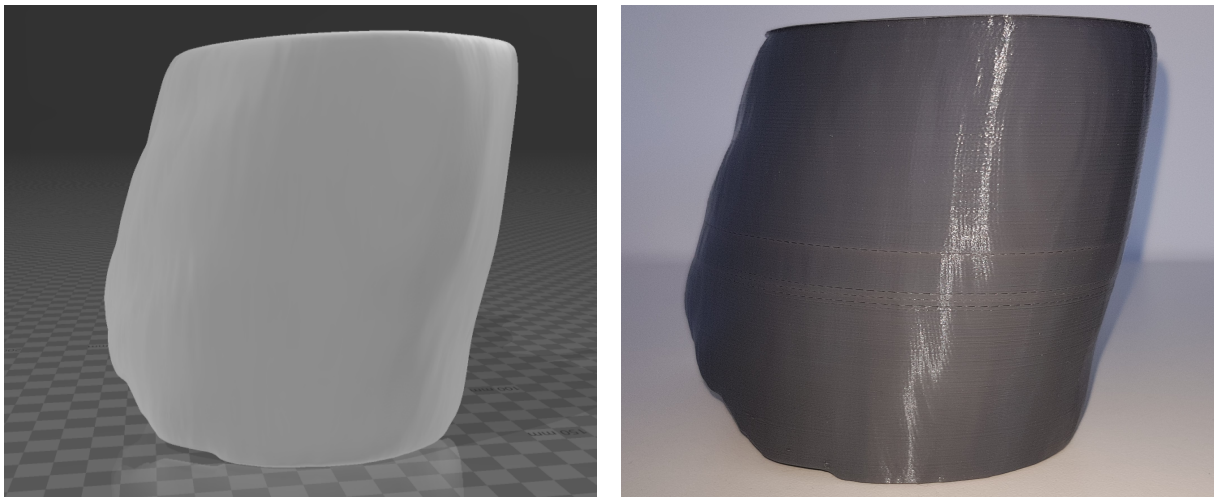


(a) Numerical Model



(b) Printed Model

Figure B.7: Comparison of the left side



(a) Numerical Model (b) Printed Model

Figure B.8: Comparison of the right side

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**