NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

EUGEN URSU

BSc in Computer Science and Engineering

PROCESSING OF LOW-CONTRAST 3D TOMOGRAPHIC IMAGES OF COMPOSITE MATERIALS

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
October, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

PROCESSING OF LOW-CONTRAST 3D TOMOGRAPHIC IMAGES OF COMPOSITE MATERIALS

EUGEN URSU

BSc in Computer Science and Engineering

Adviser: Pedro D. Medeiros
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Vasco Amaral
Associate Professor, FCT-NOVA

Rapporteur: Adriano Lopes
Assistant Professor, ISCTE

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
October, 2023

Processing of low-contrast 3D tomographic images of composite materials

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.12) [1].

Abstract

With the advancements in hardware and software technology, micro-computed tomography imaging has become an indispensable tool in materials science as it allows scientists to construct 3D images of the study materials. However, this has its disadvantages, namely the fact that it creates noise and artifacts in images, especially on low-contrast materials, making the study of the internal structure more challenging.

As such, with this thesis we intend to study innovative techniques of image processing using Python libraries with emphasis on implementations for the GPU to speed up image processing.

The results of this research will provide relevant data on the most effective methods for improving the quality of micro-computed tomography images. The thesis will contribute to the development of advanced imaging processing tools for materials science and provide a foundation for future research in this area.

Keywords: low-contrast materials, micro-CT, Image Processing, Materials Science, GPUs

Resumo

Com os avanços na tecnologia de hardware e software, a técnica de micro-tomografia computadorizada tornou-se uma ferramenta indispensável na ciência dos materiais pois permite aos cientistas construir imagens 3D dos materiais de estudo. Porém, esta apresenta desvantagens, nomeadamente o facto de criar ruído e artefatos nas imagens, sobretudo em materiais de baixo contraste, tornando o estudo da estrutura interna mais desafiante.

Assim, com esta tese pretendemos estudar técnicas inovadoras de processamento de imagens utilizando bibliotecas Python, com ênfase em implementações para o GPU, de modo a acelerar o processamento de imagens.

Os resultados desta pesquisa fornecerão dados relevantes sobre os métodos mais eficazes para melhorar a qualidade das imagens de micro-tomografia computadorizada. A tese contribuirá para o desenvolvimento de ferramentas avançadas de processamento de imagens para a ciência dos materiais e fornecerá uma base para futura investigação nesta área.

Palavras-chave: materiais de baixo contraste, micro-CT, Processamento de Imagens, Ciência dos Materiais, GPUs

Contents

List of Figures

1 Introduction

1.1 Motivation

Materials science is a field that studies the properties and characteristics of various materials, including metals, ceramics, polymers, and composites. These materials need to be represented in their three dimensions to allow the correct visualization of its constituent phases and then processed through a pipeline to a final result, which allows the study of their properties (such as micro-structures and pores) and features.

One way to build a 3D representation of the material is through micro computed tomography (micro-CT) [2], which is a non-destructive imaging technique that uses X-rays to create detailed, three-dimensional images of internal structures of small objects, such as biological samples or small mechanical parts. It is similar to medical CT (computed tomography) scans, but with a much higher resolution and smaller field of view. Micro-CT is often used to study bone structure and density, analyzing microelectronics, and inspecting materials for defects. In materials science it is used to study the structure of materials (such as grain size and crystalline structures), identify defects and variations in the material (such as cracks, pores and variations in density) and quantify the porosity to understand the materials' properties and effects. This enables us to understand the correlation between the structure and the properties of the material which can help in the design and development of new and improved materials for all kinds of applications.

However, there are some limitations. The common issues to be considered during the scan are noise and artifacts, due to the intrinsic system noises present in CT images [3]. These issues require the use of image processing techniques to properly identify the internal structure of the composite material.

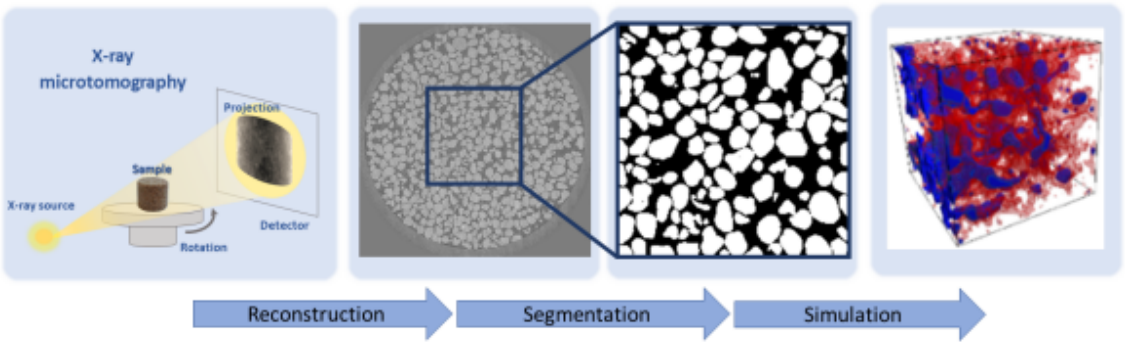


Figure 1.1: Typical workflow of composite materials processing. [4]

There are several specialized paid software such as Avizo [5] and VGSTUDIO MAX [6] that can be used to assemble the 2D CT images into a 3D volume, seen in Fig. 1.1, but free libraries such as VTK [7] and OpenCV [8] are also capable of accomplishing it.

Segmentation, seen in Fig. 1.1, is an important aspect of micro-CT research as it is a process of separating an image into different regions or segments, each of which represents a specific object or structure of interest, allowing researchers to extract meaningful information from the 3D images obtained from the scans.

In materials science this technique is used by researchers to measure the size and shape of individual grains, pores, inclusions or the volume fraction of pores and its features, allowing them to study the spatial distribution within the sample, shown in Fig. 1.1.

Segmentation can be done manually by a researcher, but often it is time-consuming and prone to errors. Therefore, automatic or semi-automatic segmentation methods are commonly used in micro-CT research. These methods use algorithms to automatically identify and isolate different regions of the microstructure, based on their gray-level, shape, or texture characteristics. Inside the segmentation pipeline, the two most used methods are thresholding which separates the image into foreground (internal structures) and background (base material) and connected component labeling (CCL) which creates a labeled image where each individual structure has a unique label assigned to it.

As many 2D CT images are necessary to create a 3D volume, the data is quite substantial in terms of size and rendering the volume requires a significant amount of computational power. For these reasons, a Graphics Processing Unit (GPU) is well suited for rendering large data, such as a one gigabyte volume, because it is designed to perform many calculations simultaneously (well suited for the parallel processing required for rendering tasks). The GPU has many cores that can perform the same operation on different data at the same time, which makes it faster than a CPU for tasks that can be parallelized.

Additionally, it is recommended the use of dedicated GPUs which are optimized for handling large amounts of data, such as images and videos. GPUs are common in rendering tasks due to their specialized memory known as VRAM (video memory). This memory is faster and has higher bandwidth than the regular system memory (RAM) which can help in processing large volumes faster.

Furthermore, for general-purpose computing on GPUs (GPGPU), many rendering software and libraries are optimized to take advantage of its parallel processing capabilities, allowing for faster and more efficient processing and rendering tasks.

Some examples are the CUDA [9] or OpenCL [10] API and a popular Python Just-In-Time (JIT) compiler Numba [11]. Numba allows the use of Python code by translating it to machine code, which can be executed directly by the CPU or GPU.

This is done by analyzing the Python code and identifying the computationally intensive parts, which are then compiled to machine code using LLVM [12], a widely used compiler infrastructure. It is also designed to work seamlessly with NumPy [13], the most widely used numerical computation library for Python, and can often provide significant performance improvements for computationally intensive tasks.

1.2 Goals

The main goal of this thesis is to process micro computed tomography (micro-CT) images to study the internal structure of low contrast composite materials, with a focus on identifying and quantifying the internal structures and pores present within the material.

The aim is to develop techniques for improving the imaging of low contrast composite materials using micro-CT and create a Python-based pipeline of processes that segment the data into their constituents by using already well defined and tested libraries. The final result would allow materials science researchers to understand how the internal structure and the presence of pores affect the properties and performance of these materials and advance their work in the materials science domain.

In particular, one of the aims of the thesis is the case where the two phases (base material and reinforcements) have similar densities thus complicating the segmentation process.

An additional goal is to implement these functionalities to allow for the use of GPU, decreasing the execution time.

1.3 Tools

In this thesis several tools will be tested to arrive at a satisfying result. The two main classes of tools under consideration are tools from the Python ecosystem and tools mostly related to GPU processing.

1.3.1 Python

Python is a powerful interpreted and dynamically-typed programming language that is widely used for a variety of applications, including scientific computing, data analysis, and machine learning. In this project, Python was selected as the main programming language due to its simplicity and readability. The Python ecosystem includes a wide

range of libraries and frameworks that can be used to accelerate the development process and many common tasks can be accomplished with minimal code, saving time and effort.

Python is an interpreted language, which means that it can be slower than compiled languages like C or C++ and as such can have performance issues for computationally intensive tasks, such as image processing or scientific simulations. However, this can be mitigated by using libraries such as NumPy and Cython, which provide a way to write code in Python that can be optimized for a near C-like performance.

Python's support for GPU acceleration is also an important consideration. There are several libraries available that allow Python code to be executed on a GPU, such as Numba [11] which was used in a previous thesis [14], CuPy [15] and PyCUDA[16]. These libraries provide a way to write code that can be executed on the GPU, significantly speeding up the execution of computationally intensive tasks such as image processing.

As such, Python is a great fit for this project, offering readability and simplicity, a wide range of libraries and frameworks and its support for GPU acceleration, making it a powerful tool for image processing. It is easy to learn and has a large community of developers who can contribute with their knowledge and experience (in the form of open-source libraries).

1.3.2 GPUs

GPUs (Graphics Processing Units) have become increasingly popular in scientific computing due to their high performance and parallel processing capabilities. Scientific computing involves performing complex mathematical computations and simulations, which can be computationally intensive tasks that require a lot of processing power. GPUs are well suited for these types of tasks because they have a large number of cores that can perform calculations in parallel, significantly decreasing the execution time. This makes GPUs ideal for tasks such as image processing, machine learning, and data analysis.

In order to take advantage of the parallel processing capabilities of GPUs, specialized programming languages and APIs such as CUDA [9] and OpenCL [10] have been developed, which allow developers to write code that can be executed on the GPU.

In summary, GPUs are well suited for scientific computing due to their high performance and parallel processing capabilities, making them a valuable tool for tasks such as numerical simulations, image processing, machine learning and data analysis.

1.4 Contributions

With this thesis we expect to produce software made in Python, where the new pipeline process used for the segmentation of the 3D volumes of low-contrast materials will offer fast and precise results. This will provide materials science researchers a better understanding of the internal structure of the materials.

It is also important to evaluate the performance of the algorithms implemented, as well as the results obtained, the latter having to be consistent with the real scanned material (no loss of information).

1.5 Chapters

The thesis will provide an overview of its goals, objectives and the methods used to achieve them. It will detail the process of development and the steps taken to implement the proposed solution, including the techniques and tools used. It will also explain the reasoning behind the chosen methods and the results obtained.

Therefore there will be an in-depth examination of the project and its outcomes, providing a comprehensive understanding of the work done and its significance to the scientific community.

Chapter 1 introduces an overview of the problem and the reasoning behind the main tools selected to solve it.

Chapter 2 dwells into the current state of the art in the field related to the proposed project. This includes a thorough review of the existing literature, as well as an analysis of the most recent and relevant research and developments. Tools such as libraries and software that could be used in the development of the project will also be mentioned.

Chapter 3 describes the devised solution discussing the several chosen methods and Python packages.

Chapter 4 shows the implementation of the solution chosen. We will achieve this by using code snippets with written step-by-step explanations, offering a thorough exposition of the developed code.

Chapter 5 covers the results obtained with their execution times and conduct a detailed statistical analysis to substantiate these findings.

Finally, chapter 6 presents a comprehensive summary of the work undertaken and the results achieved. Furthermore, we will explore potential avenues for future work.

2

Related Work

In this chapter, we will provide an introduction to micro-CT imaging and its applications, as well as an overview of the basic image processing techniques used. We will also discuss the use of parallelism and GPUs, the visualization software and Python libraries used in the field of materials science imaging. Our goal is to present a comprehensive overview of the current state of the art in micro-CT imaging and image processing and to highlight the recent advances that have greatly improved the capabilities of the technique.

2.1 Study of composite materials

As stated previously, micro-computed tomography (micro-CT) [2] is a non-destructive imaging technology that enables high-resolution, three-dimensional (3D) imaging of small samples. In recent years, the state of the art of micro-CT has advanced significantly, driven by advancements in hardware and software. Today, micro-CT systems offer higher resolution and accuracy, faster scan times, improved data processing and analysis capabilities and greater ease of use. These advancements have made micro-CT a valuable tool in fields such as materials science, biology and medicine, giving researchers insights into the internal structure and composition of materials and tissues. The technology continues to evolve with ongoing research aimed at further improving image quality, reducing radiation exposure and expanding its applications.

Micro-CT works by using X-rays to create a series of cross-sectional images of a sample. As shown in Fig. 2.1, the sample is placed on a rotating stage and X-rays are emitted from a source and pass through the sample, striking a detector on the other side. The intensity of the X-rays that pass through the sample is measured by the detector and this information is used to create an image of the sample's internal structure. The stage is then rotated to the next position and the process is repeated, producing a series of images that can be combined to form a 3D reconstruction of the sample.

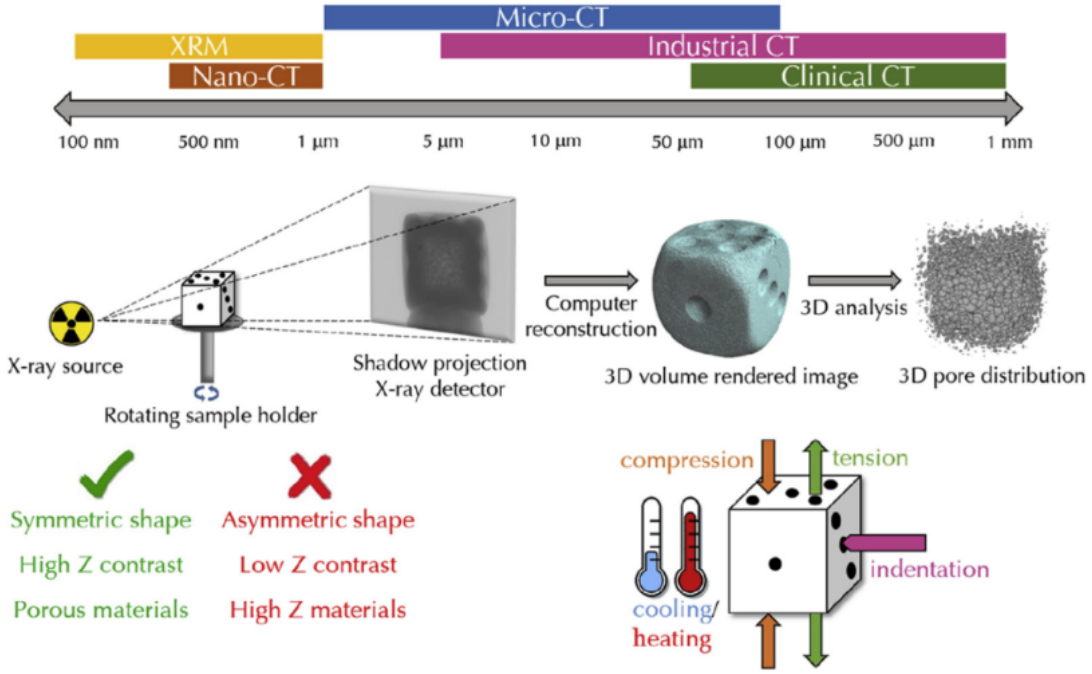
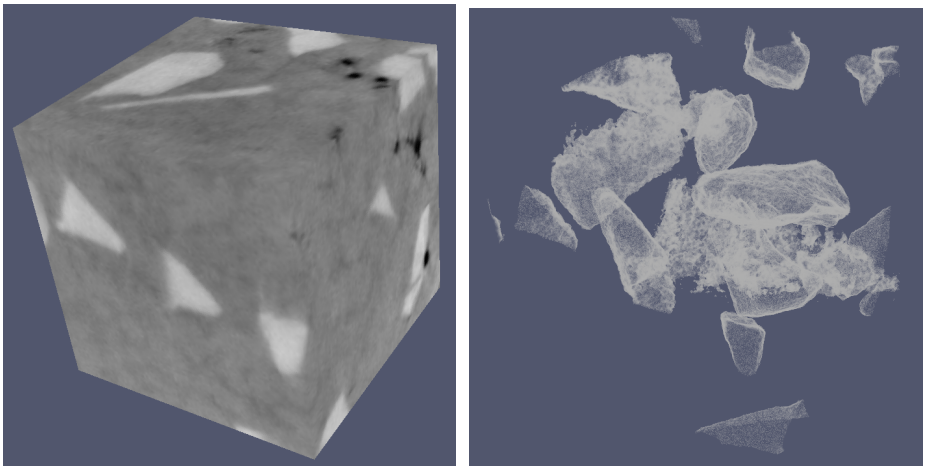


Figure 2.1: Micro-CT cheat sheet. [17]

These images are gray-scaled, where each pixel corresponds to an intensity value between 0 (lower density) and 255 (high density). There are many reconstruction algorithms and software packages that take the collection of 2D images and combine them, forming the final 3D image which shows the internal structure of the sample in high detail, allowing researchers to study its composition and properties and visualize internal features such as pores, cracks and voids.



(a) Composite material (b) Internal structure

Figure 2.2: Visualization of a composite material in Paraview. [18]

Fig.2.2 shows the internal structure of these materials which can be divided into two components: the base material (gray coloration shown in 2.2a) and the target material

(white coloration in 2.2b). The base material is the most abundant. The target material often has a higher-density, absorbing X-rays differently, and can be separated from the base material to allow the internal features to be visualized more clearly in the images.

Noise is a common issue in micro-CT imaging. Random variations of the intensity of the pixels during the micro-CT imaging process, such as photon noise or reconstruction noise, have a significant impact on the quality of the images produced.

Noise in micro-CT images can reduce the visibility of fine details, making it more difficult to study internal structures and pores. In the already generated 3D images, noise reduction techniques such as denoising filters can be used to reduce the impact of noise and the quality of micro-CT images can be improved, allowing researchers to study these features more effectively.

The NRRD (Nearly Raw Raster Data) [19] file format is a common format used to store and exchange medical and scientific imaging data, including micro-CT data. It is a plain text format that provides a simple, flexible and efficient way to store and exchange image data and metadata.

One of the key features of the NRRD file format is its ability to store multi-dimensional data, including 3D imaging modalities such as micro-CT. This allows researchers to store and exchange the entire 3D volume of data, including all of the slices, in a single file.

In addition to storing the image data, the NRRD file format also includes a header that contains information about the image, such as its size, data type and orientation. This header information is essential for interpreting the image data correctly and can be used to specify the orientation and spacing of the slices, among other things.

The NRRD file format is widely supported and has become a popular choice for researchers who need to store and exchange micro-CT data. It is easy to use and can be read by many different software packages, including open-source and commercial tools. Additionally, the NRRD format is well documented and is maintained by a community of researchers, making it a reliable and trustworthy format for storing and exchanging micro-CT data.

2.2 Previous work at FCT

There has been a lot of work done at the NOVA University of Science and Technology by the Materials Science Department as well as the Computer Science Department on the study of micro-scaled composite materials.

The Computer Science Department made Tomo-GPU [20], a software package that has enabled materials science researchers to study the composition and features of micro-scale materials by giving the package the segmentation pipeline explained later in the paper. It is also GPU implemented, allowing for faster results.

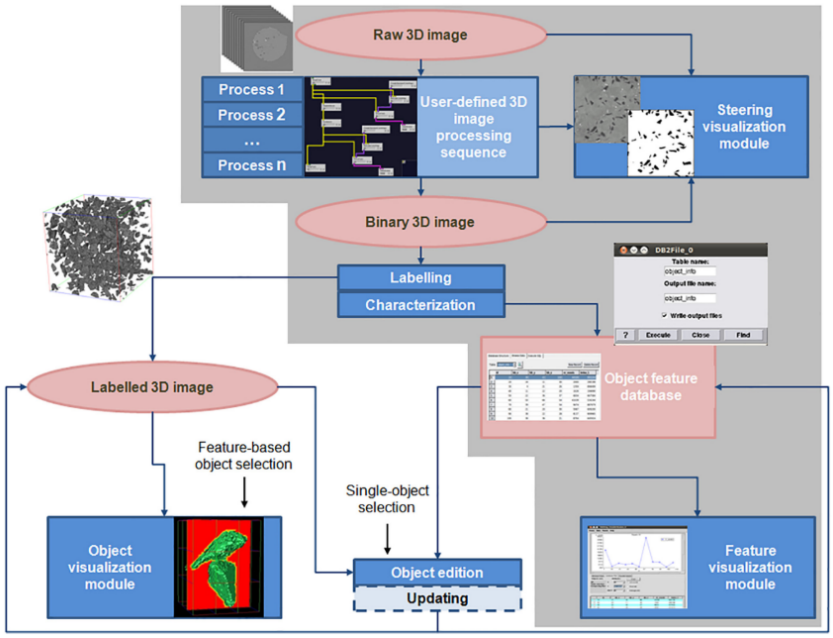


Figure 2.3: General organization of the TomoGPU system, designed for tomographic imaging processing for the purposes of material's characterization. [20]

2.2.1 High-contrast images

The Tomo-GPU software allows for the correct processing of high-contrast images, which have high variations of voxel intensity when changing from the base material to the target material. The image transformation and segmentation pipeline described in Fig. 2.3 were implemented to be used with the GPU with very good results shown in Fig. 2.4.

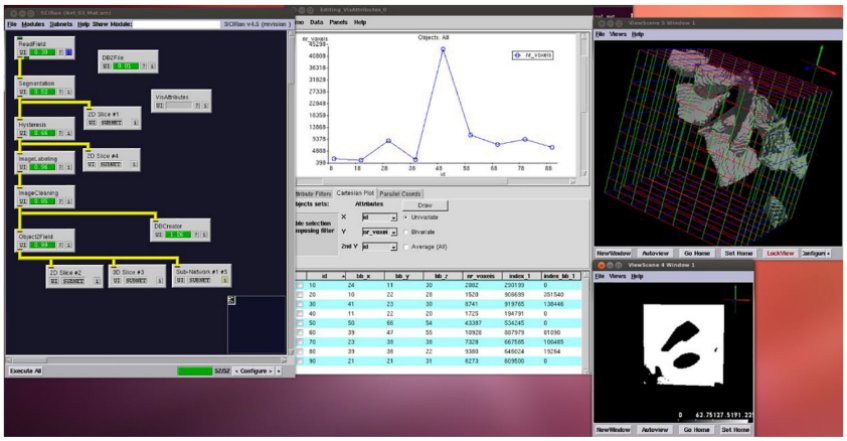


Figure 2.4: Example of binarization and segmentation of a high-contrast composite material in the Tomo-GPU environment. [20]

The description of the pipeline is based on a paper [21] which used two operations to obtain a binary image: Thresholding and Hysteresis.

The thresholding method defined by Vignoles [21] and used in previous theses [14,

22] is an histogram-based method:

$$F(x) = \begin{cases} 0 & 0 \leq x \leq min \\ x & min < x < max \\ 255 & max \leq x \leq 255 \end{cases}$$

It takes in account that there are two materials, the study material that we want to obtain and the base material, and works by giving a minimum and a maximum interval between the peaks of the histogram, obtaining the voxels that are in that interval. The remainder become either WHITE(255) or BLACK(0). This will then allow for the separation of the material with greater precision using the Hysteresis method below. This method works well for high-contrast materials as there are clear peaks in the histogram. However some particles are not filtered due to the frequency of gray levels in standard histograms not being consistent with their contributions to the representation of the image details.

After the previous step, we have a set of voxels within a range of intensity values. Some of these voxels do not belong to the particles that we want to obtain and should be attributed a new value (WHITE or BLACK).

Based on previous theses [14, 22] the Hysteresis method takes the set of voxels and for each one it will check its neighborhood voxels intensity values, attributing them a new value (WHITE or BLACK) based on the absolute majority of 0's or 255's. This method will be applied iteratively until there are no more voxel changes during a full iteration. Afterwards, for the voxels whose absolute majority was a tie (since there are 26 neighbors for each voxel), the relative majority is used. Finally, the remaining voxels are randomly assigned either WHITE or BLACK.

These binary images can then be further divided by using Connected Components Labelling algorithms, which will be explained in detail later.

Previous theses have expanded on these operations: Ribeiro [14] studied Connected Components Labelling algorithms, implemented them and compared their execution times against each other while Brito [22] used Julia language to evaluate its potential in image processing.

2.2.2 Low-contrast images

For low-contrast images the results obtained by the software in Fig.2.5 contain many imperfections/noise and the final result by the pipeline is considered unsatisfactory.

Therefore there is a need to create new algorithms for the pipeline that can deal with the small differences in voxel intensity, as well as the noise generated by the micro-CT scan.

Quaresma et al. [24] uses a complex method to filter the noise and artifacts in the 3D images, which involves the removal of the pores inside the composite material as well as the increase in contrast between the base material and the target material, among others.

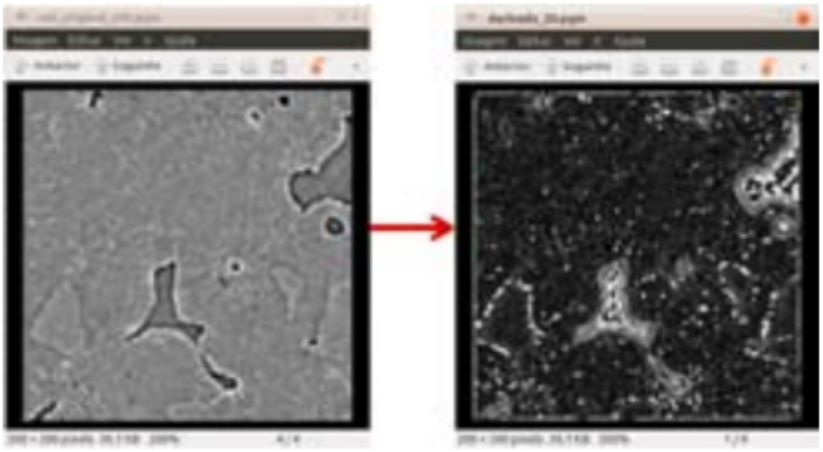


Figure 2.5: Example of edge detection of a low-contrast composite material in the Tomo-GPU environment. [23]

2.3 3D image processing

Image processing refers to the techniques used to manipulate or transform images to enhance their quality, extract information from them or convert them into a format that is more suitable for analysis, storage or dissemination. It involves applying various algorithms and techniques to analyze, filter and modify images for various applications, such as medical imaging, remote sensing, computer vision and more.

It plays a crucial role in materials science by enabling researchers to analyze and understand the properties and behavior of their materials.

2.3.1 Operations

In general, image processing operations can be categorized into four types [25]:

- Pixel operations: the output for each pixel is solely determined by its corresponding input, without any influence from other pixels within the same image. A common example of a pixel operation is thresholding, which transforms input pixels above a certain threshold to white and the rest to black. Other pixel operations include modifying brightness and contrast, inverting the image, and applying log and power law transformations;

- Local (neighborhood) operations: the output for each pixel considers not just the input at a single pixel, but also the surrounding pixels in determining the output. This allows for a more nuanced processing of the image. Examples of local operations include edge detection, smoothing filters such as the average and median filters, and sharpening filters like the Laplacian and gradient filters. These operations can also be adaptive, meaning that the results can vary depending on the specific pixel values in each region of the image;

- Geometric operations: uses specific geometric transformations to determine the output of a pixel based on the input levels of selected pixels, rather than all the pixels in the image. This sets them apart from global operations, which consider the input of all pixels. Examples of geometric operations include scaling, rotation, and translation of the image;

- Global operations: involves the utilization of information from all the pixels in the image to determine the output for a single pixel. This output may be based on the pixel values themselves, or it may be based on statistics calculated from all the pixels. A distance transformation, which assigns the minimum distance between an object pixel and the background pixels, is a classic example of a global operation. Other examples include histogram equalization/specification, image warping, Hough transforms, and connected components analysis.

There is a need to study the best image processing operations to use in this thesis as previous operations have not been proven satisfactory.

2.3.1.1 Thresholding

Thresholding is a technique used in image processing to convert a grayscale image into a binary image (black and white image) based on a threshold value. Typically, a threshold value is selected either manually or automatically so that all pixels in the image with values above the threshold are set to white (or 255) and pixels with values below the threshold are set to black (or 0).

As such, thresholding often is used in image segmentation to separate the objects of interest from the background based on their intensity values.

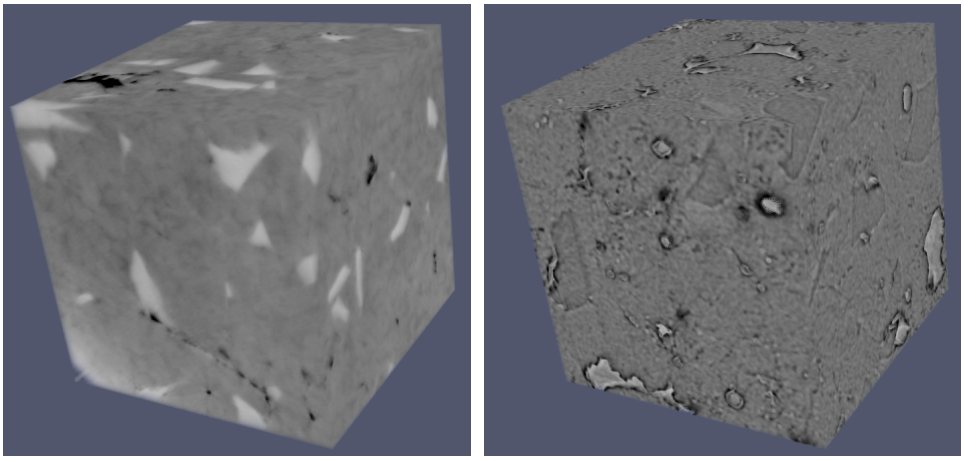
There are different categories of thresholding methods [26], to name a few:

- histogram shape-based methods, where the peaks, valleys and curvatures of the smoothed histogram are analyzed;

- clustering-based methods, where the gray-level samples are clustered in two parts as background and foreground (object) or alternately are modeled as a mixture of two Gaussians;

- entropy-based methods result in algorithms that use the entropy of the foreground and background regions, the cross-entropy between the original and binarized image.

Popular methods such as Otsu's method [27], which is an automatic clustering-based thresholding method that minimizes the weighted sum of within-class variances of the foreground and background pixels to establish an optimum threshold, or Shanbang's entropy-based thresholding method that considers the fuzzy memberships as an indication



(a) High-contrast material (b) Low-contrast material

Figure 2.6: Visualization of two composite materials in Paraview. [18]

of how strongly a gray value belongs to the background or to the foreground [26] have been created and used in various scientific fields.

This work focused on low-contrast materials which have a small difference in their physical or chemical properties when compared to their background or surrounding material. These materials can be difficult to detect or distinguish from their background using traditional imaging techniques, such as the micro-computed tomography (micro-CT) used. This can make it challenging to identify and analyze these materials.

To improve the segmentation of the internal structures of low-contrast materials shown in Fig. 2.6b, various image enhancement techniques can be used that are already integrated in several Python libraries, such as histogram equalization, contrast stretching and several of the most popular thresholding methods such as the previously stated ones.

Thresholding methods, such as the gray-level information histogram, in which the amplitudes of its components are cumulative summations of weighted statistical values of the gray levels in different regions [28] or the two-stage adaptive binarization approach for the segmentation of noisy low-contrast images based on Sauvola's binarization algorithm which minimizes the over-segmentation, or omission of weak signals in the foreground, without significantly increasing the number of irrelevant objects [29].

Peter et al. [30] shows promising results of segmentation based on watershed of low contrast structures in bone micro-CT images.

Various image processing techniques were used by Ortalan et al. [31] on low-contrast material to enhance the contrast and define the correct shape and size of structural features with minimal artifacts.

Al-Ameen et al. [32], created an innovative technique for contrast enhancement of CT images using normalized gamma-corrected contrast-limited adaptive histogram equalization.

Bhattad, Willson, and Thompson [33] use a novel method for the segmentation of

low-contrast three-phase X-ray computed tomography images of porous media with good results.

Varfolomeev, Yakimchuk, and Safonov [4] created a solution based on convolutional neural networks (CNN) architectures for segmentation of 3D microtomographic images.

These are specifically used to deal with low-contrast material and could be a base for the implementation of the thresholding method used for this thesis.

2.3.1.2 Hysteresis

Hysteresis [34] can be useful for images that have a high level of noise or for images that have intensity variations within the object of interest. It can help to maintain the continuity of the object and eliminate false detection of noise as an object. It is a popular technique in edge detection, object segmentation and image binarization. The method is computationally efficient and can produce good results when the intensity range of the object of interest is very different from the background intensity values.

We are working with low-contrast composite materials where the base and target material have similar intensities, meaning that the previous thresholding method cannot be used as there no clear defined peaks in the histogram to binarize the 3D image. As such, this previously studied method is insufficient and other alternatives should be considered.

2.3.1.3 Connected Component Labeling

Connected Component Labeling (CCL) [35] is an intermediate level algorithm used in image processing and computer vision to identify and label the connected regions of an image. The goal of CCL is to group together all the pixels in an image that are connected to each other, such as pixels that belong to the same object or region of interest. The basic idea behind CCL is to iterate through the pixels of an image and for each pixel that has not been labeled, perform a search of its surrounding pixels to find all connected pixels that have the same value. These connected pixels are then assigned the same unique label and the process is repeated until all pixels have been labeled.

It is a fundamental technique in image processing, used in a wide range of applications including object recognition, image segmentation and image analysis.

Various CCL algorithms have been implemented over the years and can be classified as [36]:

Label Propagation algorithms

Semi-supervised learning techniques used to assign class labels to unlabeled data points in a dataset. The labeled data points serve as "seeds," and the unlabeled data points are labeled based on their similarity to the labeled data. There are various methods to implement label propagation, such as spreading the labels based on similarity or using a graph-based approach where the data points are represented as nodes and the labels are propagated through the edges. However, since these

algorithms do not access the image in a regular pattern, they are not well-suited for parallel or hardware-based implementation;

Label Equivalence algorithms

Technique that seeks to find a set of equivalent labels for a given set of labels in a dataset. The algorithm does this by finding a mapping function that transforms the original labels into a new set of labels, which offer the same classification performance as the original set. The objective of these algorithms is to identify a new set of labels that are more compact, interpretable, or robust than the original labels. Given that they process pixels in a raster scan order, they are well-suited for parallelization and hardware-based implementation with streamed image data.

The CCL algorithms can also be categorized by the number of scans/passes until convergence (final image), such as one-scan, two-scan or multi-scan algorithms.

The algorithms can be further divided into [36]:

Pixel-based algorithms

They work by iterating through each pixel in an image and analyzing its neighboring pixels to determine if it is part of a connected region of pixels. If a pixel is found to be part of a connected region, it is assigned a label and then the algorithm proceeds to analyze the neighboring pixels of the labeled pixel. This process is repeated until all pixels in the image have been analyzed and labeled.

Run-based algorithms

They scan the image from top to bottom and from left to right, and for each run of pixels, assign a label to it and check if the run is connected to any previously labeled runs. If so, it is assigned the same label as the previously labeled run, if not, it is assigned a new label. This process is repeated for all the runs of pixels in the image.

Block-based algorithms

They divide the image into small blocks and then analyze each block separately. For each block, a run-based algorithm is applied to identify and label the connected regions of pixels. Then the algorithm proceeds to analyze the neighboring blocks to check if they have any regions that are connected to the regions in the current block. This process is repeated until all the blocks have been analyzed and labeled.

Previous work by Ribeiro [14] compared three CCL algorithms and their implementations on a GPU: ACCL (run-based multi-scan algorithm) and BUF & BKE (block-based two-scan algorithms). The results of their work showed that ACCL demonstrated superior performance in almost every case, except for high-lighting run-based algorithm deficiencies.

2.3.2 CPU algorithms

For Thresholding, most of the algorithms and methods in section 2.3.1.1 are CPU algorithms.

For CCL, new algorithms have been published, promising faster execution times such as:

A pixel-based algorithm [37] that uses two different masks for processing two different types of object voxels. One type of mask is used when the voxel preceding the object voxel being processed is an object voxel, and the other type is used otherwise. In either case, an optimal order is used for checking the voxels in the corresponding mask.

LSL3D [38], a run-based algorithm that combines an unification approach based on a finite state machine to improve its efficiency on simple images and a double-line mechanism and computational reuse for complex ones. On top of a scalar extension, SIMD (SSE4, AVX2, AVX512) instructions are used to accelerate the RLE compression and decompression steps.

As the focus is on the binarization of the composite materials, an implementation of a newer CCL algorithm is not considered and the ACCL algorithm already studied will be used.

2.3.3 CPU libraries

As this thesis will use Python as the main language, there are several libraries that deal with image processing:

OpenCV [8] is an open-source computer vision and machine learning software library. Some of the features provided by OpenCV include image processing, video analysis, object detection and machine learning. The caveat is that it is not designed to deal with volume data (3D) right away (needs to process each 2D slice).

SciPy [39] is an open-source Python library for scientific computing. It is built on top of the NumPy library and provides a wide range of functionality for tasks such as image processing, signal processing, interpolation and more. The "scipy.ndimage" module provides a variety of image processing functions such as filtering, thresholding and segmentation. However, scipy is not as comprehensive as libraries such as OpenCV and scikit-image, which are more specialized and have more advanced image processing capabilities.

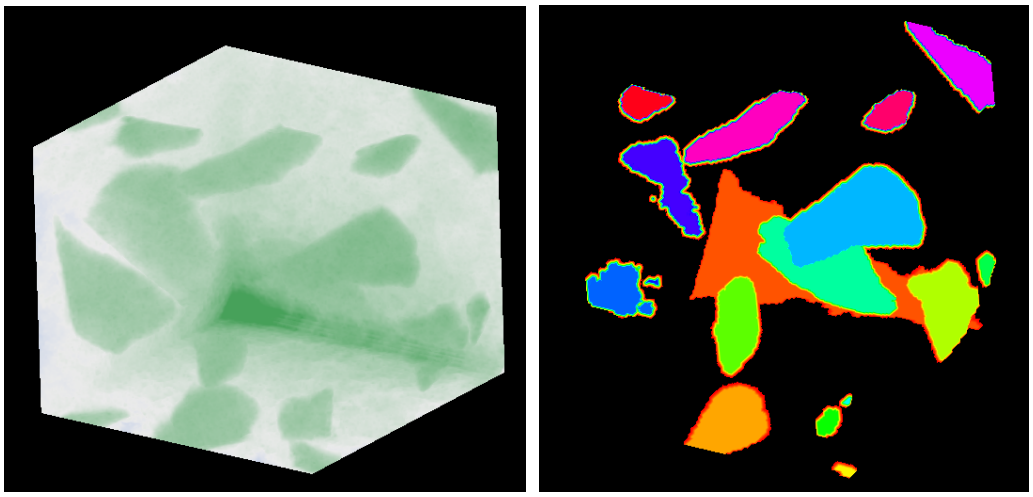
scikit-image [40] (skimage) is an open-source python-based image processing library that is built on top of other popular libraries such as NumPy and SciPy and has some parts written in Cython (programming language which is a superset of Python programming language designed to have performance like C programming language) to achieve good performance. It provides a wide range of image processing algorithms, including image filtering, geometrical transformations and feature detection. It has a clean, well-documented and easy-to-use API, which makes it a good choice for beginners and experts alike. Additionally, it has built-in support for multi-dimensional image processing,

making it easy to work with images of any dimensionality. It is also actively maintained and new features and algorithms are regularly added. While OpenCV and SciPy have GPU implementations, skimage only has GPU implementation for a limited number of types of files (does not have for the NRRD file type which is used for this thesis).

2.3.4 Experimentation with high-contrast images

A past project [14] was studied in detail to better understand how to segment and label the study material of the 3D volumes provided beforehand. This allowed an in-depth study of the techniques used, the correct visualization of the 3D volumes as well as the Python library Numba, speeding up the processing and subsequent output of the results. All of this was made to run on the CPU to simplify the problem.

After that, several Python libraries were tested using their own image processing methods to segment the 3D volumes and compare the results with each other and the method stated earlier. The library scikit-image proved to have all the methods necessary to the pipeline and the time taken was decremented due to the efficient use of Numpy arrays during the thresholding and the connected component labeling steps, as shown in Fig. 2.7.



(a) Original 3D image (b) Labeled reinforcements

Figure 2.7: Visualization of reinforcements obtained from thresholding and CCL in napari. [41]

Processing methods of some software packages were tested. The ones that stood out were Fiji/ImageJ, where the functions where already implemented with easy results although with some difficulty due to the use of the CPU, and Paraview, due to the inbuilt functions in its interface as well as the automatic use of the GPU allowed a fast inquiry of the results expected, shown in section 2.1 in Fig. 2.2.

2.4 GPUs

The use of GPUs (Graphics Processing Units) in research has been growing in recent years and they have proven to be especially useful in the field of micro-CT imaging. GPUs are specialized hardware designed to perform complex calculations rapidly and efficiently, making them well suited for tasks such as image processing and analysis.

Nvidia GPUs are among the most widely used GPUs for research and they have been used in a variety of applications, including micro-CT imaging [42]. Nvidia GPUs are known for their high performance, versatility and support for advanced technologies, such as CUDA (Compute Unified Device Architecture), facilitating their use by researchers.

In the field of micro-CT imaging, GPUs can be used to perform a variety of tasks, including image processing, segmentation, and visualization [43]. For example, GPUs can be used to perform image processing tasks such as filtering, denoising and correction, which are essential for obtaining high-quality images. They can also be used to perform segmentation tasks, such as identifying and isolating different structures within an image, and to perform visualization tasks, such as generating 3D images and animations.

In addition to the speed and efficiency provided by GPUs, they also allow researchers to perform complex calculations in parallel, which can greatly reduce the time required to analyze large datasets. This is particularly important in micro-CT imaging, where datasets can be very large and complex and the analysis can be time-consuming.

2.4.1 GPU algorithms

For thresholding, Prahara et al. [44] created parallel implementations on GPU for three adaptive image thresholding methods (Otsu, ISODATA, and minimum cross-entropy) achieving 4-6 times faster results. Najafi et al. [45] proposed three CUDA GPU parallel implementations of the Nick local image thresholding algorithm for faster binarization of large images.

Even though these are used for 2D images, they can be adapted or used as reference for an implementation of a 3D thresholding method.

For CCL, newer algorithms emerged like SAUF4 1BPP [46] or a very recent one by Zhao et al. [37].

2.4.2 GPU libraries

As the work will be heavily done on GPU implementation, each of the libraries has its corresponding GPU implementations.

OpenCV [8] has CUDA modules that can be added to the installation of the library. Although, for CCL, so far, the only available CUDA algorithm is Block-Based Komura Equivalence (BKE) [46], which was already studied in previous work [14] and it is generally slower than the ACCL algorithm [14].

SciPy [39] is compatible with CuPy [15], an open-source array library for GPU-accelerated computing with Python. CuPy utilizes CUDA Toolkit libraries to make full use of the GPU architecture. In most cases, it is as easy as replacing scipy with cupyx.scipy in the Python code.

Skimage [40] can use cuCIM [47], an open-source, accelerated computer vision and image processing software library for multidimensional images used in biomedical, geospatial, materials and life science and remote sensing use cases. Unfortunately, it does not support NRRD files.

CLIJ2 [48] is a GPU-accelerated image processing library for ImageJ/Fiji, Icy, Matlab and Java. It provides hundreds of operations for filtering, binarizing, labeling, measuring in images, projections, transformations and mathematical operations for images.

2.5 Visualization tools

There are several tools that can be used to visualize and process 3D images:

- napari [41], a fast, interactive, multi-dimensional image viewer for Python. It is designed for browsing, annotating and analyzing large multi-dimensional images. It is built on top of Qt (for the GUI), vispy (for performant GPU-based rendering) and the scientific Python stack (numpy, scipy). It includes critical viewer features out-of-the-box, such as support for large multi-dimensional data, layering and annotation. The py-clEsperanto-assistant is a yet experimental napari plugin for building GPU-accelerated image processing workflows.
- Fiji [49] is an image processing package—a “batteries-included” distribution of ImageJ2, bundling a lot of plugins, to facilitate scientific image analysis. It has 3D support, CLIJ2 and clEsperanto available as an update site.
- Paraview [18] is an open-source multiple-platform application for interactive, scientific visualization. It has built-in extensions for writing Python scripts (although somewhat restrictive for its own functions) and GPU support (also has Nvidia IndeX which allows the use of a cluster of GPUs to process the data).

The one that ended up being chosen was the napari visualization tool, as it has all the necessary functions for this thesis, is a "plug and play" library which can be used directly and has some interesting plug-ins which will be described further on.

It is very easy to use:

```
1 import napari
2
3 viewer = napari.Viewer()
4 viewer.add_image(image)
5 napari.run()
```

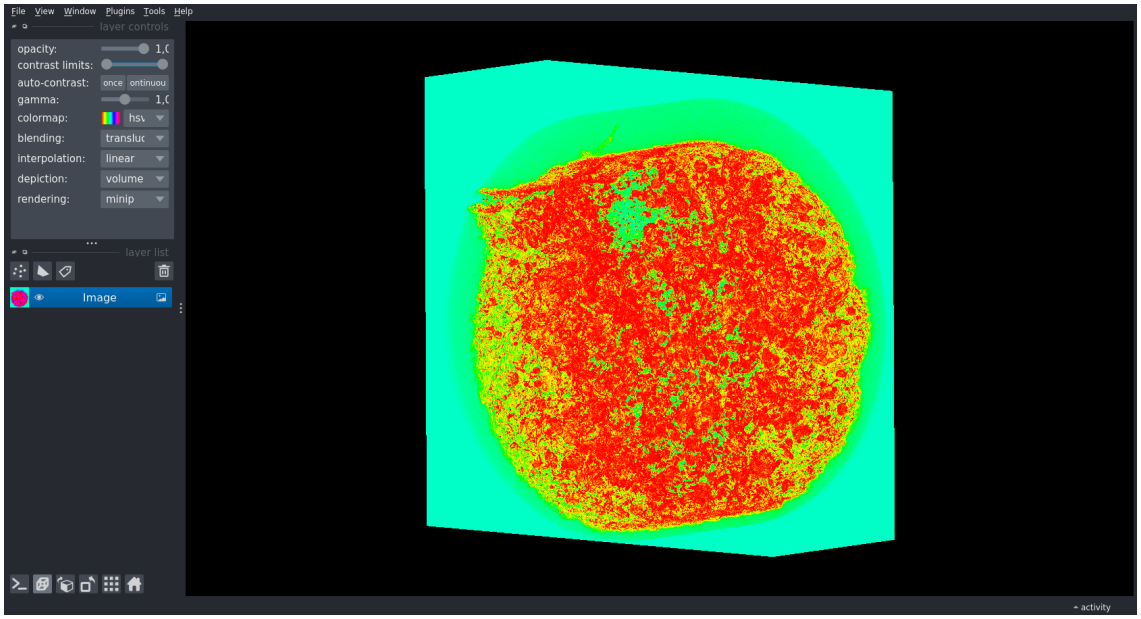


Figure 2.8: Napari Viewer with a loaded 3D image.

3

Devised solution for low-contrast images

This chapter offers an in-depth examination of the internal structure of the studied composite material, the procedures to be used as well as the libraries that hold them.

In this chapter we will show a more theoretical approach to the procedures and algorithms used to better understand how they function. Additionally, practical examples demonstrating the application of the selected libraries will be presented, offering a hands-on perspective.

3.1 Organization

To accomplish what is proposed, it is necessary to divide the processing pipeline into two main steps. The first one consists in identifying the pores in the sample in order to obtain a mask of the 3D image where one equals a pore and zero is everything else. This will then allow for further processing through the labeling of each pore and individual study of its characteristics or by separating them into different categories, namely the ones mentioned in section 3.2.3.

The second stage of the process is identifying the reinforcements described in section 3.2.2. This is an essential step after the identification of the pores since the solution implemented separates all the structures from the base material without identifying which are from each category.

Other steps are done during the processing pipeline of the low-contrast images, namely pre-processing and post-processing, which will be extensively covered in the next chapter.

It is important to highlight that some parts of the processing pipeline are done on the CPU, such as I/O operations. However the major bulk of the operations rely on GPU acceleration due to the substantial size of the 3D image volume and the computationally intensive nature of the operations being executed.

The methods used during the development of the processing pipeline will be described in the next chapter.

3.2 Composite Sample Analysis

Composite Sample analysis is important as it is the cornerstone for understanding its internal structures.

This thesis focuses on the analysis of a cylinder-shaped composite material with dimensions 432x1024x1024 (Z,Y,X axis), shown in Fig.2.8. Since the internal structure is harder to visualize in 3D images, going forward, most figures are 2D image slices of the 3D images obtained using the napari visualizer.

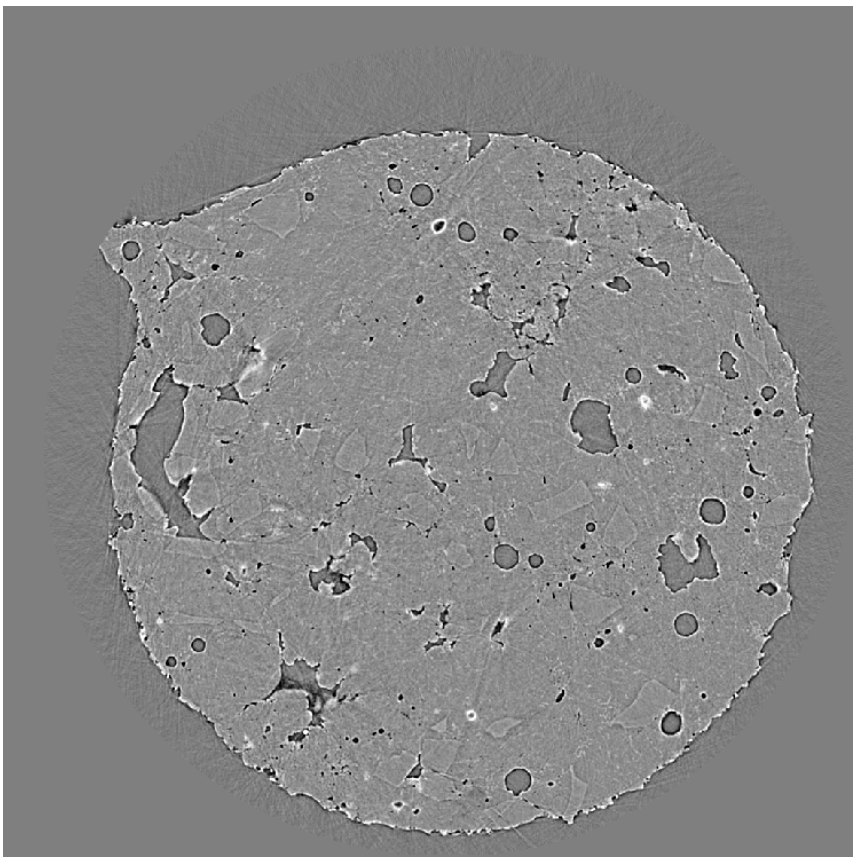


Figure 3.1: 2D Slice of the sample.

This sample comprises of tungsten carbide reinforcements of various shapes and sizes embedded within a metal matrix base made out of aluminum[24]. Additionally, it contains two types of pores which naturally occur during the creation process: spherical, formed from small pockets of gas, and irregular, formed due to metal contraction.

3.2.1 Base material

Given that the base material is very similar in density to the reinforcements we can effectively consider them equivalent in intensity.

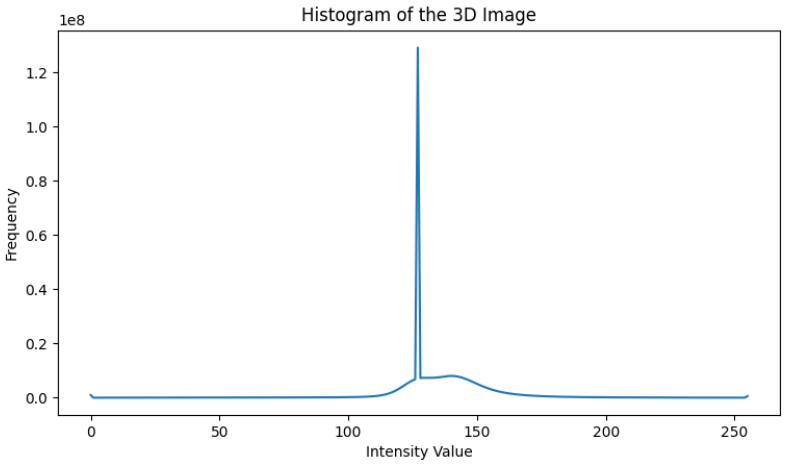


Figure 3.2: Histogram of the sample.

Fig.3.2 is a representation of the histogram of the intensity values of the sample. The peak seen is regarding the intensity value of the border of the sample and can be excluded. The intensity values of the reinforcements and the base material do not have clear peaks. Hence, this sample can be considered a low-contrast composite material and therefore more difficult to differentiate its internal structure.

3.2.2 Reinforcements

The reinforcements are small structures with a median diameter of 37 micrometers that make up 10% of the total volume of the sample. Previous collected information show a cone-shaped convex geometry of its structures with some being fragmented[24] likely due to pore infiltration and artifacts.

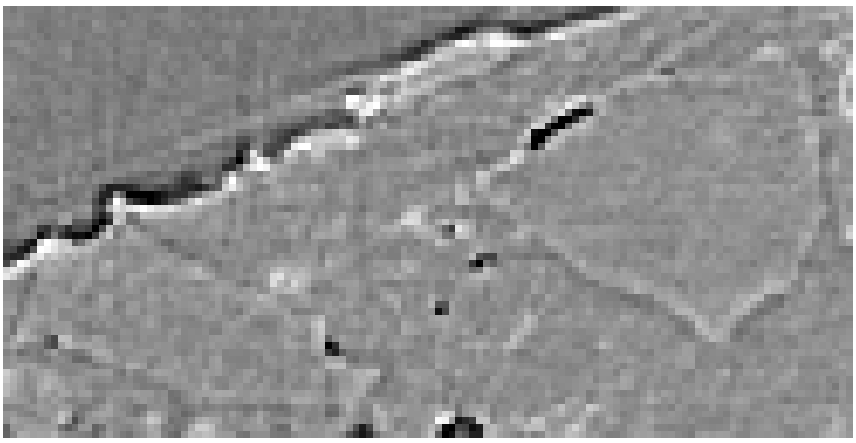


Figure 3.3: Example of 2 reinforcements in the sample.

3.2.3 Pores

Pores are pockets of air formed during the creation process of the sample. They can come in several types and sizes, most commonly, the ones explained below.

Spherical

Made by existing packets of gaseous phases that remain in the sample. Smaller in size and with a high degree of sphericity.

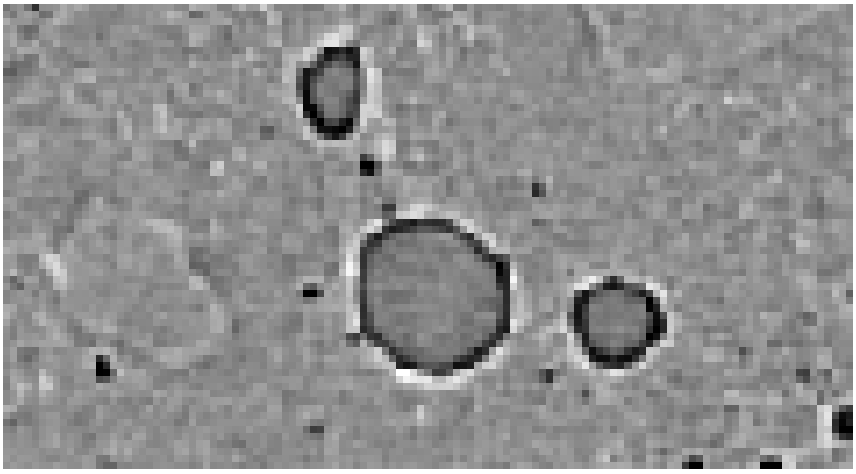


Figure 3.4: Example of 3 spherical pores in the sample.

Irregular

Made by contraction of the metal matrix during the creation process. Bigger in size with irregular shapes.

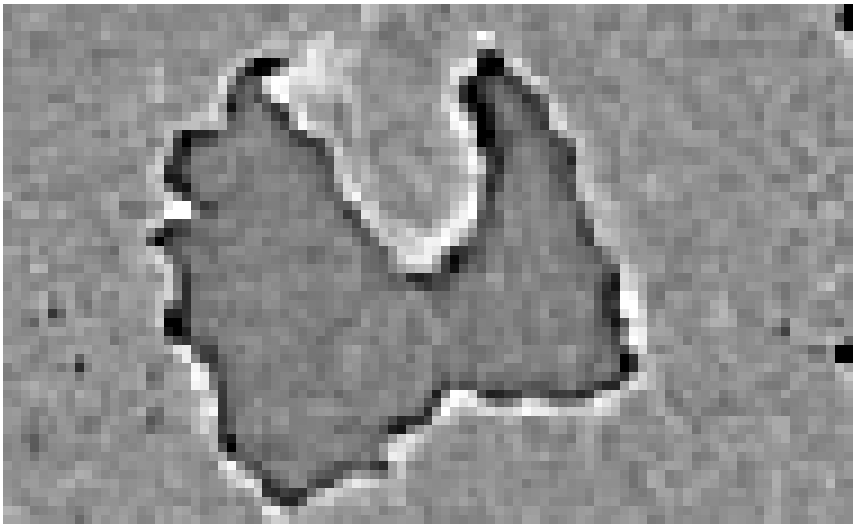


Figure 3.5: Example of an irregular pore in the sample.

3.2.4 Artifacts

Artifacts in images are undesired, unintended features or distortions that are introduced during image acquisition, processing, or transmission. These artifacts can degrade image quality, affect interpretation, and impact the effectiveness of various image analysis tasks.

This section provides an overview of the most common artifacts encountered in the 3D image:

Gaussian noise [50]

Random variations in pixel values following a Gaussian or normal distribution is a common form of noise that can significantly degrade image quality. This noise appears as a grainy or speckled pattern in the image and can obscure important details, making accurate interpretation and analysis challenging. To mitigate the adverse effects of Gaussian noise, various image filtering techniques are employed, the most common one being the Gaussian filter. Median or uniform filters are also used. These filtering techniques play a vital role in enhancing image quality and ensuring that images are more suitable for accurate analysis.

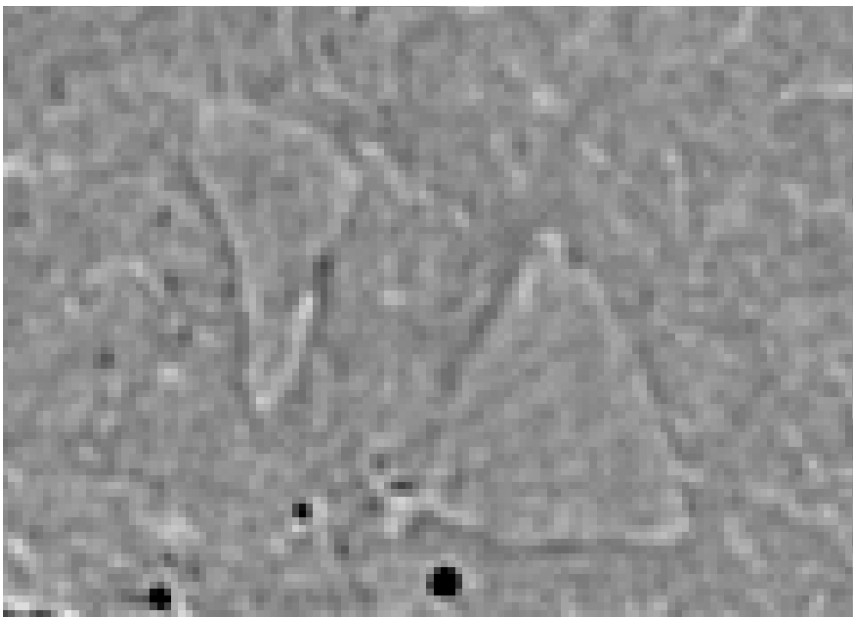


Figure 3.6: Example of gaussian noise in the sample.

Ring artifacts [51]

Dark or light unwanted circular patterns that can appear in CT images, normally created due to a miss-calibrated detector. When one or more detector elements do not function correctly or produce inconsistent signals, it leads to variations in the intensities of the sample data. These variations can manifest as concentric rings in the final image, degrading its overall quality.[3] Efforts are continually made to minimize and correct these artifacts through advances in imaging technology and image processing techniques.

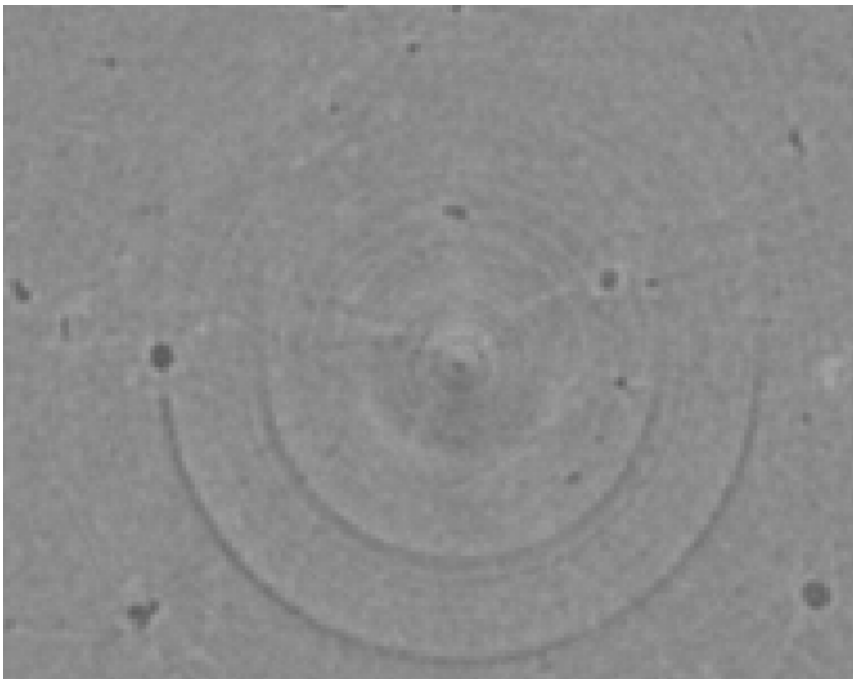


Figure 3.7: Example of rings in the sample.

Beam-hardening artifacts [51]

Cross or star-like patterns that occurs when the X-ray beam is filtered differently as it passes through materials of varying densities causing streaks and lines to radiate outward from high-attenuation structures. It can lead to misleading information about the composition or density of the imaged sample and it adversely affects the accuracy of segmentation, the process of distinguishing different materials based on their intensity differences in the image. These issues collectively reduce the precision of analytical tasks aimed at separating materials by their varying intensities, such as thresholding and segmentation.

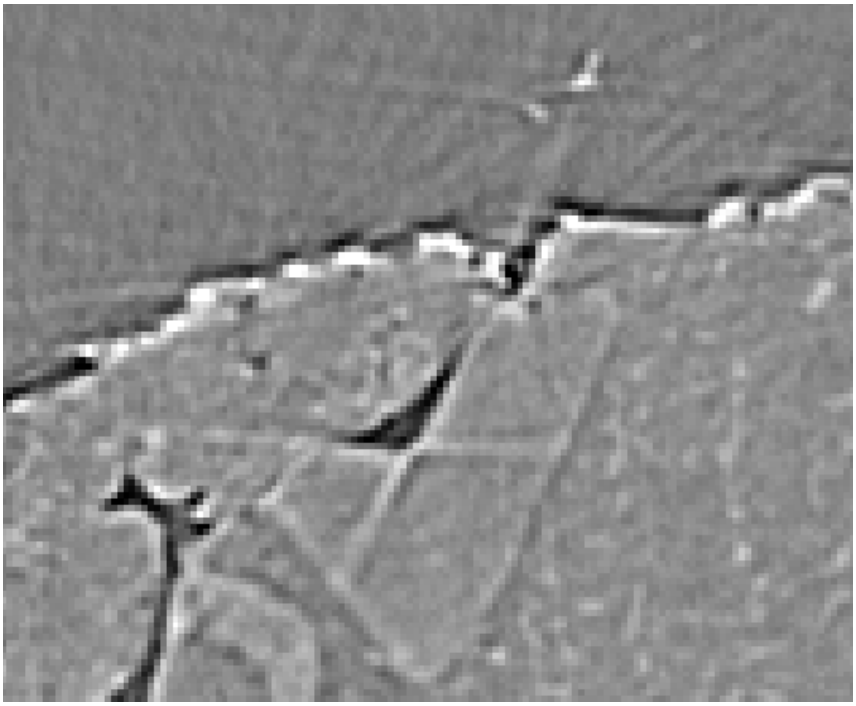


Figure 3.8: Example of beam-hardening in the sample.

3.3 Processing Pipeline

A processing pipeline in image processing is a structured sequence of interconnected steps that serves as a systematic and modular framework for the transformation, analysis, and extraction of valuable information from raw images. Each stage in the pipeline performs a specific operation on the input, and the output from one stage becomes the input for the next. By breaking down the complex process of image analysis into well-defined steps, these pipelines enhance efficiency, reproducibility, and scalability of image processing tasks across various domains.

3.3.1 Pre-processing

The pre-processing pipeline consists of two steps: 3D volume creation and contrast stretching.

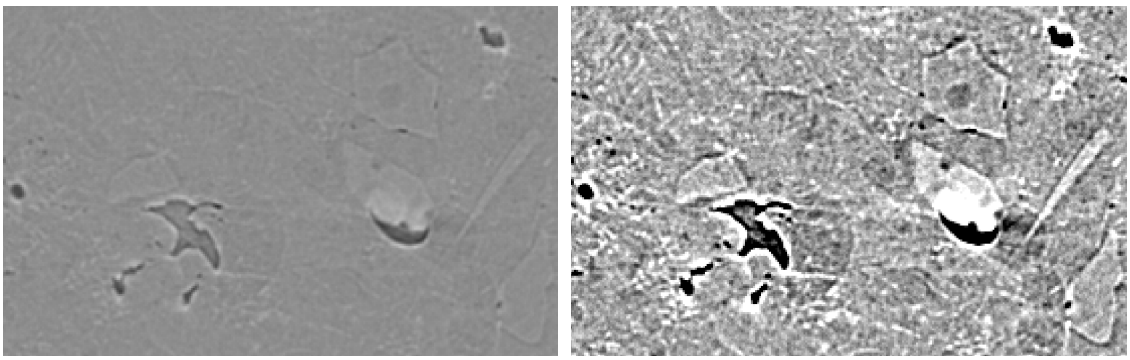
The first step involves reading and writing the .raw sample data file into an useful 3D NumPy array that can be processed by the libraries to be used. The information is not shown correctly yet (it is a list of values) since we do not know the shape of the array. Therefore we must read a .vol.info file, which has the dimensions of the 3D volume, and afterwards reshape the NumPy array into a viewable 3D volume.

Parts of the 3D volume of the micro-CT sample obtained lose contrast between its components in its first and last slices (Z-axis), making the image more blurry and challenging the visualization of its details. Hence, the second step consists of contrast stretching [52] (often called normalization) which is a digital image processing technique used to

improve the visibility of details in an image by expanding the range of pixel intensity values. This can be done by applying the following function:

$$F(x) = \begin{cases} min & 0 \leq x \leq min \\ ((x - min)/(max - min)) * 255 & min < x < max \\ max & max \leq x \leq 255 \end{cases}$$

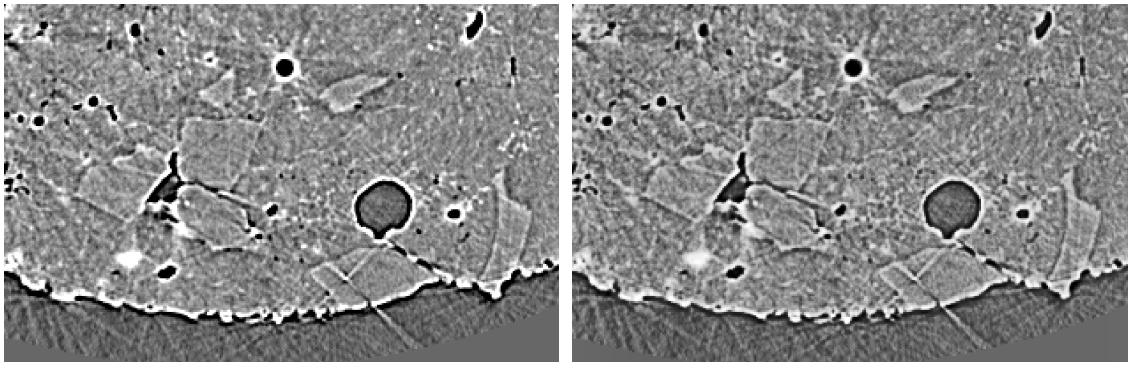
With this function we can take a lower and higher bound of the voxel values and apply a linear transformation to map the original pixel intensities to the new range. This function is more robust than the classic normalization as a single outlying pixel with either a very high or very low value can severely affect the minimum and maximum values and lead to very unrepresentative scaling. Hence we can select the 2th and 98th percentile in the histogram (that is, 2% of the pixels in the histogram will have values lower than the minimum, and 2% of the pixels will have values higher than the maximum).



(a) Before contrast stretching (b) After contrast stretching

Figure 3.9: Visualization of contrast stretching on the sample.

Contrast stretching can be applied to the hole image, yet its caveat is that it can amplify some characteristics (mostly bright spots). Thus it should only be used small percentages to not exaggerate its characteristics. There are other more sophisticated methods such as contrast limited adaptive histogram equalization (CLAHE)[53]. However, the slower processing does not translate into any additional benefit to the final image. As we can see in the Fig.3.10, the final product of the two algorithms is very similar, the stark difference being that CLAHE attenuates the brighter spots in the image.



(a) After contrast stretching (b) After CLAHE

Figure 3.10: Contrast stretching versus CLAHE.

3.3.2 Pore identification

The identification of the pores in the 3D volume is simple as there is a stark contrast in the voxel values between them and the base material. Their edges are also more pronounced where the values are closer to zero (black). The sample 3D image has a lot of noise generated during the creation of the micro-CT. Therefore we need to reduce it beforehand. We start by applying a smoothing filter to reduce noise and to preserve more prominent features such as edges. The Gaussian filter is a widely used image processing technique employed to reduce noise in an image while preserving its overall structure. It is commonly used as a pre-processing step in various computer vision and image processing tasks, including feature extraction, object recognition, and image segmentation. It was selected in this case because it reduces noise while preserving the edges of the various structures inside the image better than other smoothing techniques, such as simple averaging or median filtering.

The Gaussian filter [54] operates on an input image, denoted as $I(x, y)$, and produces an output image, denoted as $G(x, y)$, where (x, y) represents the spatial coordinates of a pixel in the image. The formula for applying a 2D Gaussian filter to an image is as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} * I(x, y)$$

Where:

- G(x, y) is the pixel value in the output (filtered) image at coordinates (x, y).
- I(x, y) is the pixel value in the input image at coordinates (x, y).
- σ is the standard deviation of the Gaussian distribution, which controls the extent of smoothing. A larger σ results in more extensive smoothing.

The formula convolves the input image with a 3D Gaussian kernel, which is represented by the Gaussian function $e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$. This kernel emphasizes the center pixel while gradually

attenuating the influence of neighboring pixels as they move farther from the center. This attenuation has a smoothing effect while preserving the overall image structure.

Next we applied a thresholding technique to divide the 3D image into regions of interest. In this case, the 3D image is binarized, meaning that the objects we want to extract (pores) are assigned the value one (foreground) and the rest of the image is assigned the value zero (background). The chosen method is Otsu's thresholding[27], also known as Otsu's method or Otsu's binarization, which is a nonparametric and unsupervised method widely used in image processing to automatically determine an optimal threshold value that allows conversion of selected segments in a grayscale image into a binary image. The process begins by computing the histogram of the image to capture the distribution of pixel intensities and then normalizing it into a probability distribution. Subsequently, the cumulative distribution is calculated and the mean intensity of the entire image is determined. The key idea behind Otsu's method is to maximize the between-class variance while minimizing the within-class variance for various threshold values.

In this thesis one of our goals was to automatize the processing pipeline. To accomplish this we selected Otsu's method over other thresholding methods, as this is the most popular automatic binarization method that calculates the optimal threshold of the image to divide it into foreground and background. Using this method we obtain the pores of the 3D image in a fragmented state due to the edges of the original sample having some noise or artifacts created during the micro-CT process. Hence, we apply some dilation and erosion[55] to smooth and bind the edges of the pores.

Dilation is a morphological operation that expands the regions of foreground pixels in a binary image. It works by placing a structuring element (a small binary mask or kernel) centered on each pixel in the image and setting the pixel at the center of the structuring element to 1 (foreground) if at least one pixel under the structuring element is 1. Dilation is useful for tasks like object grouping, shape analysis, and feature extraction. It makes objects thicker and can be used to fill small gaps in the foreground.

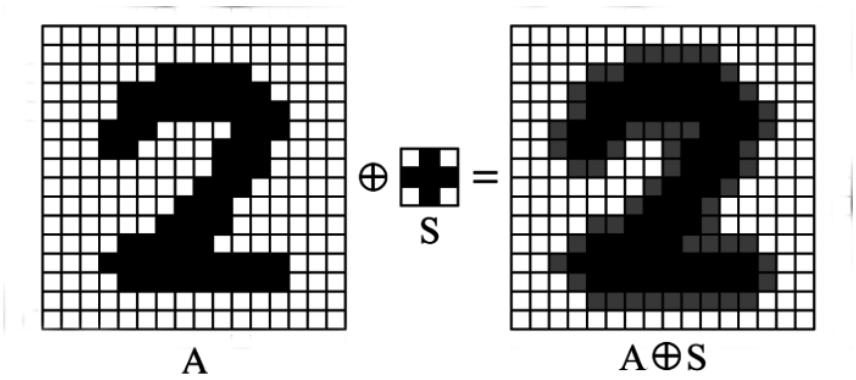


Figure 3.11: Example of dilation of image A by the structuring element S. [56]

Erosion, on the other hand, shrinks the regions of foreground pixels in a binary image. It works by placing a structuring element on each pixel in the image and setting the

pixel at the center of the structuring element to 0 (background) if all the pixels under the structuring element are 1. Erosion is valuable for tasks like noise reduction, separating connected objects, and obtaining the skeleton of objects. It makes objects thinner and can remove small protrusions.

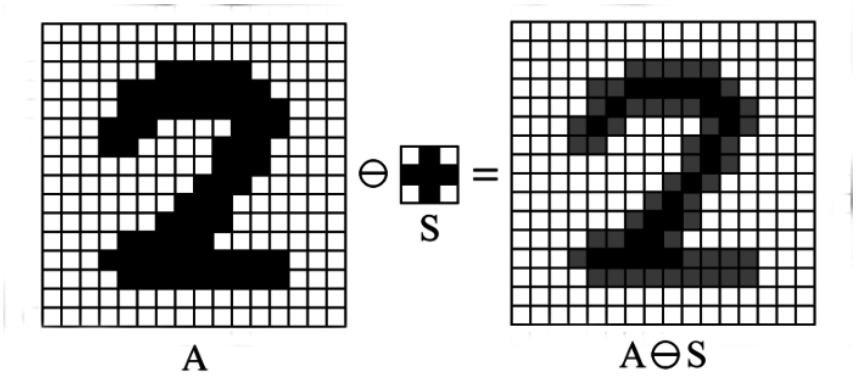


Figure 3.12: Example of erosion of image A by the structuring element S. [56]

Both binary dilation and erosion are fundamental building blocks for more complex morphological operations like opening (erosion followed by dilation) and closing (dilation followed by erosion). These operations play a crucial role in image processing tasks such as segmentation, feature extraction, and noise reduction, allowing for improved analysis and understanding of binary images.

Afterwards we considered darker spots on the image as pores, which were then added to the final result using a simple threshold. At the end of this step, we have obtained a mask of the pores in the sample which will be used on other steps further on. As a final step, we apply a stronger binary opening to detach the pores from the border for further analysis.

3.3.3 Reinforcement identification

Identifying the reinforcements is harder than identifying the pores because there is little to no difference in intensity values between the first ones and the base material. However, a way to distinguish the reinforcements is by using the borders between them which we call edges. These edges have very similar intensity to the materials and can contain holes and noise which impairs the correct segmentation by the traditional thresholding techniques.

Firstly, we must address the noise present in the 3D sample. Thus we chose the Chambolle's Total Variation (TV) regularization method[57] as it stands out as an influential approach known for its efficiency and effectiveness in addressing various image denoising and restoration problems. Is is designed to minimize the total variation of an image, effectively reducing noise while preserving important structural details. In its essence it minimizes the following function:

$$E(u) = \frac{1}{2}\|u - f\|_2^2 + \lambda \cdot \mathrm{TV}(u)$$

Where:

- u is the image estimate.
- f is the observed noisy image.
- TV(u) represents the total variation of the image u.
- λ is the regularization parameter that controls the trade-off between data fidelity and regularization.

Chambolle's TV regularization method offers several advantages as it converges to a solution efficiently, effectively removes noise while preserving edges and important image features, is versatile and can be applied to various image processing tasks. However, it also has some limitations where tuning the regularization parameter (λ) can be challenging, the method may struggle with certain types of noise and textures and it might require a good initial estimate for optimal results.

It differs from the previous stated method, the Gaussian filter also used for noise reduction, by more effectively preserving the edges and fine details of the image.

Next we use Otsu's method to binarize the 3D image in order to obtain a binary array where the reinforcements and part of the base material are foreground. This circumvents one of the previous problems in an earlier iteration where we only obtained the edges which we then had to fill with a computationally expensive algorithm (Scipy's *binary_fill_holes()*).

Parts of the reinforcements are attached to the base material due to the difficulty of distinguishing their edges. As such, we apply the opening morphological operation to detach each internal structure from each other. Afterwards we label the image and then erase the bigger connected components, corresponding to parts of the base material. Doing this will ensure that only the reinforcements remain.

3.3.4 Post-processing

Post-processing involves using the results obtained from the last two steps and applying operations that allow for further examination of the 3D image. This usually involves labeling each individual object so that we can later obtain its properties, be it a reinforcement or a pore. This is achieved by using a labeling algorithm, our options being choosing either one already created in a previous thesis[14] or one available in Scipy's ndimage library. The later was the preferred choice due to its efficiency (linear time complexity), its memory optimization (union-find strategy), and ease of use. This labeled array can be further analysed and characterized by using the cuCIM function *measure.regionprops()* which obtains several properties of each object, such as area, diameter, volume, among others, all of which can later help us in the evaluation chapter.

These properties can also facilitate the execution of other operations, namely dividing the pores into categories, as stated before, or showing the size distribution of reinforcements in the sample.

3.4 Summary Diagram

In conclusion, we processed low-contrast images using several image processing methods to enhance the internal structures visibility. Subsequently, segmentation algorithms were employed to accurately delineate distinct objects or regions within the images.

The comprehensive methodology and steps undertaken in this process are represented in a detailed diagram. This diagram serves as a visual representation of the segmentation workflow and offers a clear depiction of the process undertaken to address the challenges posed by low-contrast images.

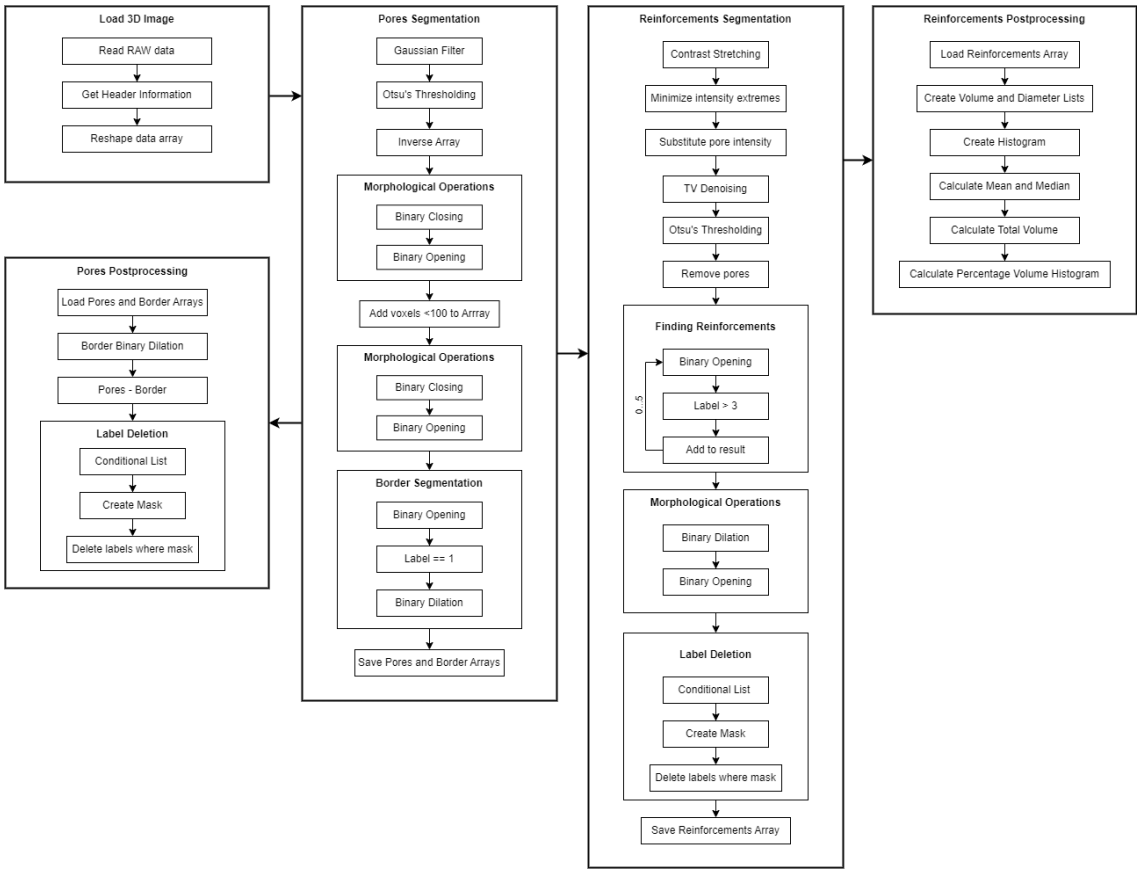


Figure 3.13: Diagram of the processing pipeline.

3.5 Software Packages

In the previous chapter, sections 2.3.3 and 2.4.2, we described several software packages used in image processing that were potential tools to be used in this thesis, while also highlighting their main advantages and disadvantages.

In this section we will present a brief overview of the software packages used to develop the solution, the reasoning behind their selection as well as an example of how to use each one.

CuPy

A Python library designed for high-performance numerical computing that extends the capabilities of NumPy by leveraging the power of NVIDIA GPUs. It provides efficient interface for executing array operations on GPUs, significantly accelerating computations for tasks like matrix operations, deep learning, and scientific simulations. CuPy mirrors the NumPy API, making it easier for users familiar with NumPy to transition to GPU-accelerated computing without major code modifications.

```python
import numpy as np
import cupy as cp

cpu_array = np.array([1, 2, 3])   # create a simple array
gpu_array = cp.asarray(cpu_array) # moves the array to the GPU device
gpu_array = cp.mean(gpu_array)    # result is 2.0
cpu_array = cp.asnumpy(gpu_array) # move the array to the CPU host. gpu_array.get() is an
↪  equivalent function
```

Figure 3.14: Example of use of the NumPy and CuPy libraries.

As we have shown, CuPy is an intuitive tool, most of the time being used as a drop-in replacement by exchanging the NumPy package with the CuPy one.

Scipy's ndimage

A powerful Python library for multidimensional image processing and signal analysis. It provides a wide range of functions and tools for working with n-dimensional arrays, commonly used in tasks like image manipulation, filtering, feature extraction, and more. This module is especially valuable in scientific and medical image analysis, where it allows for advanced operations such as convolution, morphological operations, and measurements on n-dimensional data. It integrates with NumPy, making it convenient for users to process and analyze complex data sets efficiently.

```python
import cupy as cp
from scipy import datasets

GPU = True

if GPU:
    from cupyx.scipy import ndimage as ndi
    image = cp.array(datasets.ascent())
else:
    from scipy import ndimage as ndi
    image = datasets.ascent()

result = ndi.gaussian_filter(image, sigma = 1)
```

Figure 3.15: Example of use of the CuPy and SciPy libraries.

This example shows a simple use of a Gaussian filter on an image either by using the CPU or the GPU, highlighting Scipy's ndimage library's ease of use.

cuCIM

A open-source GPU-accelerated computer vision and image processing software Python library for multidimensional images. It serves as a crucial tool for scientific and medical imaging applications, deep learning, and computer vision. This API mirrors the scikit-image but does not have all its functions implemented and it is interoperable with CuPy making its integration simple. Unfortunately this package is only available for Linux distributions.

```
1  import cupy as cp
2  from skimage.data import astronaut
3
4  GPU = True
5
6  if GPU:
7      from cucim.skimage import filters
8      image = cp.array(astronaut())
9  else:
10     from skimage import filters
11     image = astronaut()
12
13 result = filters.gaussian(image, sigma = 1)
```

Figure 3.16: Example of use of the CuPy and SciPy libraries.

The example above shows the ease of use of the cuCIM package.

napari

An open-source Python library for interactive and multi-dimensional image visualization and exploration. It's designed to simplify the process of visualizing complex image data and enables users to explore and analyze large datasets in a user-friendly interface. Napari supports various data types, including 2D, 3D, and multi-channel images, as well as points, labels, and shapes, making it a valuable tool for researchers in fields such as biology, neuroscience, and materials science. It integrates with popular scientific libraries like NumPy, Scikit-image, and Dask, allowing for efficient data manipulation and analysis. It also allows for the visualization of every step in a pipeline by sending the trasformed images to the visualizer.

```python
import cupy as cp
import napari
from skimage.data import astronaut
from cucim.skimage import filters



viewer = napari.Viewer()

image = cp.array(astronaut())
result = filters.gaussian(image, sigma = 1)
viewer.add_image(cp.asnumpy(image))

napari.run()
```

Figure 3.17: Example of interaction between all the chosen libraries.

This example shows the use of the napari visualizer. The image is sent to the GPU where the Gaussian filter is applied. Afterwards the image is sent back to the CPU for visualization in the napari environment.

3.6 Closing remarks

In conclusion, this chapter has outlined a two-stage processing pipeline essential for the successful segmentation and analysis of the 3D image.

The upcoming chapter will delve into the methods and techniques employed during the development of the processing pipeline. The chapter will include code snippets accompanied by step-by-step explanations with the corresponding image results.

4

Implementation of the segmentation pipeline

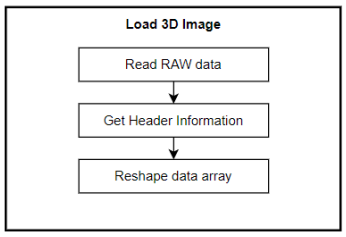
This chapter provides an in-depth discussion of the implementation process behind this thesis, wherein we delve into the technical details of how we achieved the crucial task of segmenting 3D images of low-contrast samples into its internal structures and its study thereafter. In many instances, composite materials exhibit subtle variations in contrast between the reinforcement phases and the surrounding matrix as well as a lot of noise which makes the task more difficult. Furthermore, in the realm of academic research, the accurate segmentation of low-contrast samples opens doors to deeper investigations into material behavior under various conditions. Researchers can explore the effects of different reinforcement configurations, manufacturing techniques, and environmental factors, yielding valuable insights that drive innovation and advance the frontiers of materials science.

Throughout this chapter, we will elucidate the key components of our implementation, including the choice of imaging techniques, the selection of suitable software tools, and the development of custom algorithms. We will also discuss the challenges encountered during the implementation and the strategies employed to address them. Our aim is to provide a comprehensive and replicable framework that can be applied to similar research endeavors in the field of materials science and engineering.

4.1 Pre-processing

In the pre-processing section of our implementation, it is essential to prepare the 3D image data for segmentation in order to enhance its quality and ensure accurate results. The sample data's folder contains two files: the RAW volume data and the header containing the dimension of the volume.

Below, we detail the actions performed using the following diagram and code snippet:




```
import auxiliary as aux
import numpy as np

# Load the image from a file
file = ''ME-216 - SiC FGMMC/C27/RAW/c27_12_2_256'
header = aux.open_info_file('{}.vol.info'.format(file[:file.rfind("_")]))
image = np.fromfile('{}.raw'.format(file),
↪  dtype=np.ubyte).reshape([header['NUM_Z'],header['NUM_X'],header['NUM_Y']])
```

- Extract Header Information: We start by reading the header file *.vol.info*. It contains the dimensions (number of voxels along each axis) of the 3D image which is essential metadata for the next step. In the code snippet, we utilize the *aux.open_info_file()* function to open and read the header file associated with the 3D image data. The header information is stored in the header variable, which allows us to access important parameters for the subsequent processing step.

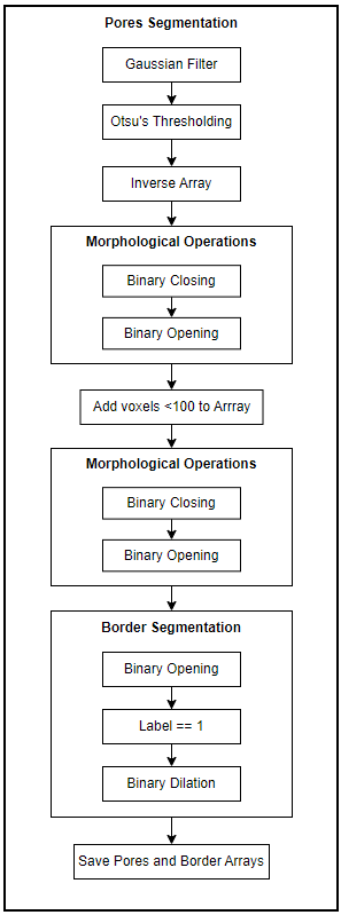
```
def open_info_file(filename):
    # Open the file
    with open(filename, 'r') as file:
        lines = file.readlines()

    # Process the lines and split into a dictionary
    info_dict = {}
    for line in lines:
        if line.startswith('NUM_'):
            key, value = line.split('=')
            key = key.strip()
            value = int(value.strip())
            info_dict[key] = value

    return info_dict
```

- Loading the 3D image: The variable file contains the path to the RAW data file, which represents the composite material sample. The data in this file is stored as a 1D array of unsigned bytes (*dtype=np.ubyte*). The *np.fromfile()* function reads this binary data and subsequently reshapes it into a 3D array based on the information provided by the header file from the step above.

By loading the 3D image data and extracting the header information, the pre-processing section establishes the foundational parameters required for segmentation and analysis steps. The accurate representation of the physical dimensions and the organization of the image data are essential for conducting research into the structure of the composite material.

4.2 Pore identification

In this step we employ a series of image processing techniques to identify and segment the pores within the 3D image data. Below, we outline the steps taken to accomplish pore identification:

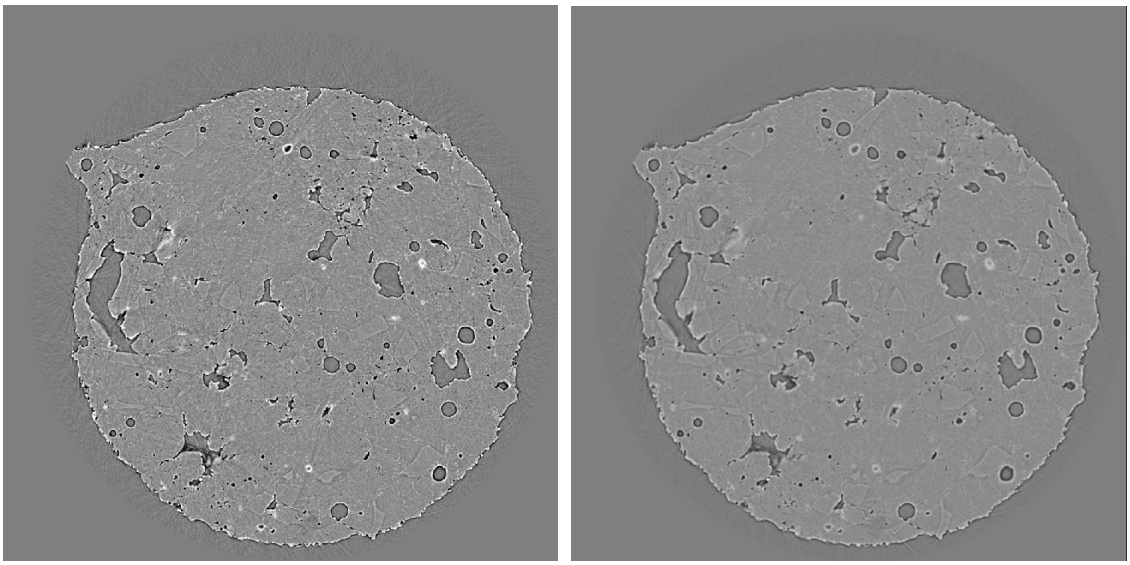


The following code implements the figure above:

```
image = cp.asarray(image, dtype=np.ubyte)
pores = image.copy()

pores = ndi.gaussian_filter(pores, sigma = 1)
viewer.add_image(cp.asnumpy(pores), name = 'gauss')

```

```
pores = pores > filters.threshold_otsu(pores)
pores = util.invert(pores)
viewer.add_image(cp.asnumpy(pores), name = 'thresh+inv')

pores = ndi.binary_closing(pores, structure=morphology.ball(1))
pores = ndi.binary_opening(pores, structure=morphology.ball(2))
viewer.add_image(cp.asnumpy(pores), name = 'morphops')

pores[image < 100] = 1
pores = ndi.binary_closing(pores, structure=morphology.ball(2))
pores = ndi.binary_opening(pores, structure=morphology.ball(1))
viewer.add_image(cp.asnumpy(pores), name = 'addsmall')

pores2 = ndi.binary_opening(pores, structure=morphology.ball(3))
pores2 = measure.label(pores2)
border = (pores2 == 1)

pores = cp.asnumpy(pores).astype(np.byte)
border = cp.asnumpy(border).astype(np.byte)
```

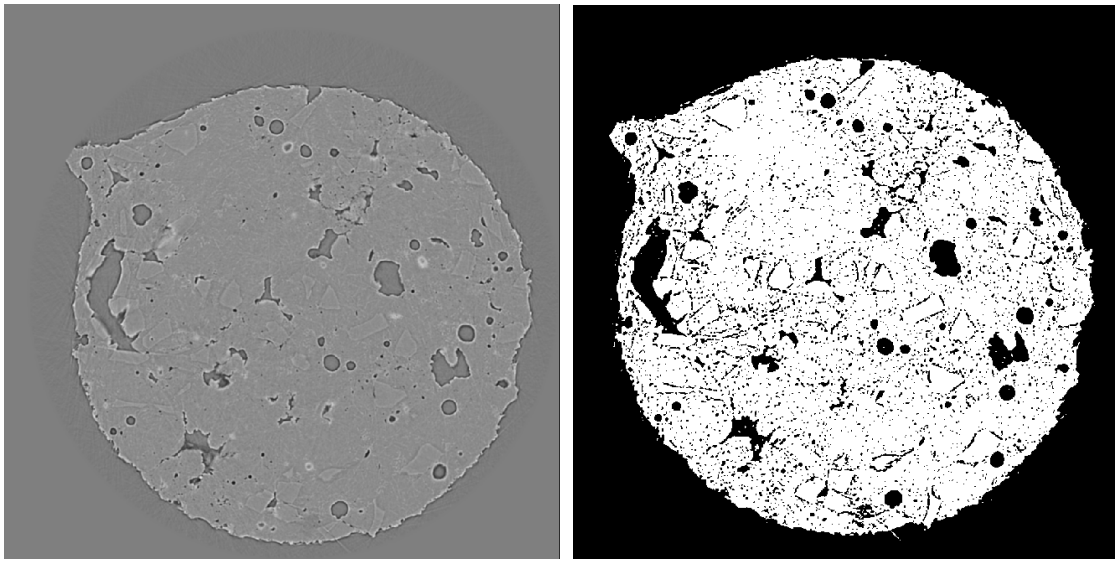
- Data Preparation: Initially, we create a copy of the 3D image loaded in the pre-processing stage and store it in the variable *pores*. This copy ensures that the original data remains intact throughout the pore identification process.
- Gaussian Smoothing: To enhance the image quality and reduce noise, we apply a Gaussian filter to the sample using the *ndi.gaussian_filter()* function. Smoothing helps in creating a more continuous and well-defined image. After thorough testing, the sigma parameter set to one gives the 3D image a good degree of smoothing.



(a) Original 3D image (b) Gaussian Smoothing

Figure 4.1: Application of the Gaussian filter to the 3D image.

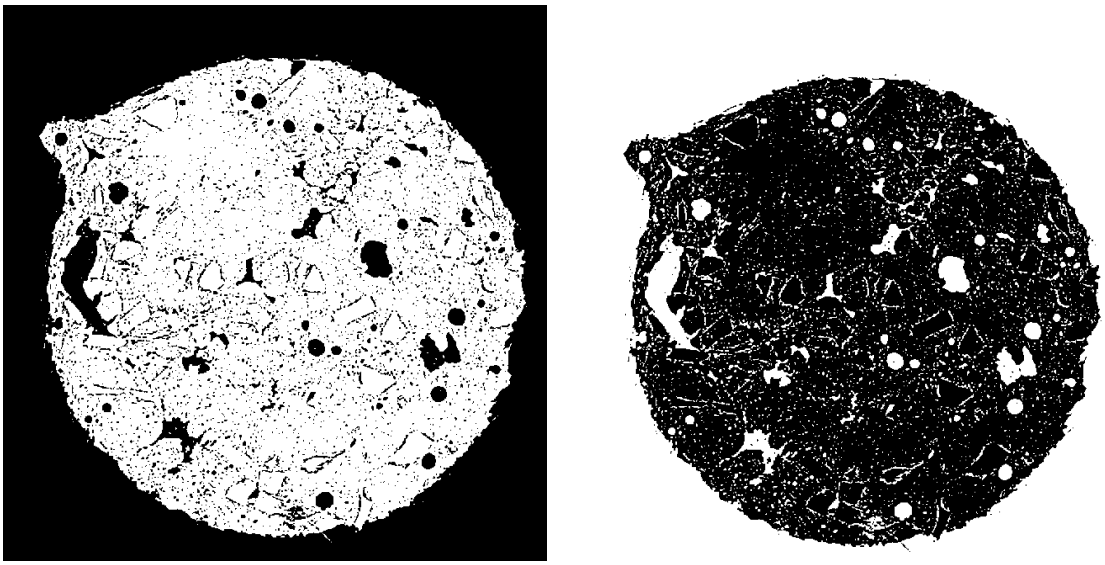
- Thresholding with Otsu's Method: We perform image thresholding using Otsu's method, which automatically calculates an optimal threshold value to distinguish between pores and other materials. The resulting binary image shows a good distinction between the pores which are black (zero) and the reinforcements and base material which are white (one). This thresholding step essentially segments the pores from the rest of the material. We can also see fragmented edges of the reinforcements which shows the difficulty to separate them from the base material.



(a) Gaussian Smoothing (b) Otsu's Thresholding

Figure 4.2: Application of Otsu's method to the 3D image.

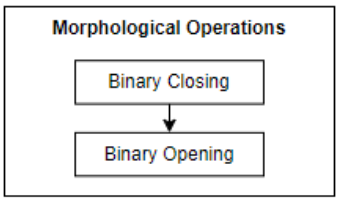
- Inversion: In the binary image obtained from thresholding, we invert the values so that pores become foreground objects, while the rest of the material becomes background. We also see that the edges of the reinforcements are also considered foreground and as such there is a need to erase them.



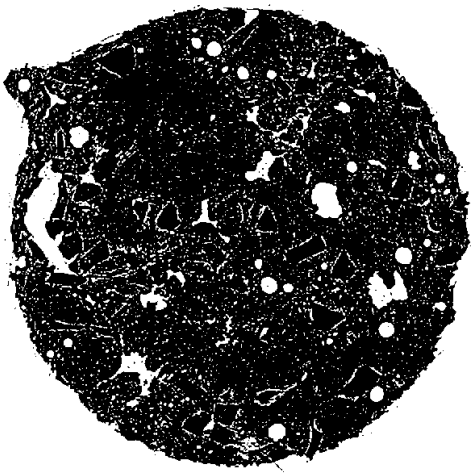
(a) Otsu's Thresholding (b) Inverted image

Figure 4.3: Application of inversion of the Otsu's Thresholding 3D image.

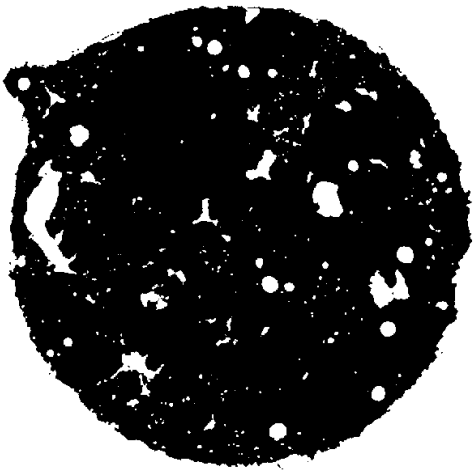
- Morphological Operations: We apply morphological operations to further refine the pore segmentation. First, we perform binary closing using a spherical structuring element (*morphology.ball(1)*) to close small gaps in the pore regions. Then, we apply binary opening with a slightly larger structuring element (*morphology.ball(2)*) to remove the small edges of reinforcements and smooth the pore boundaries. These operations help in achieving a more accurate and connected representation of the pores.



(a) Step



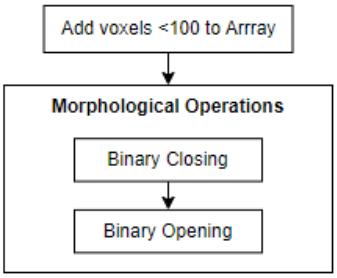
(b) Pores binary image



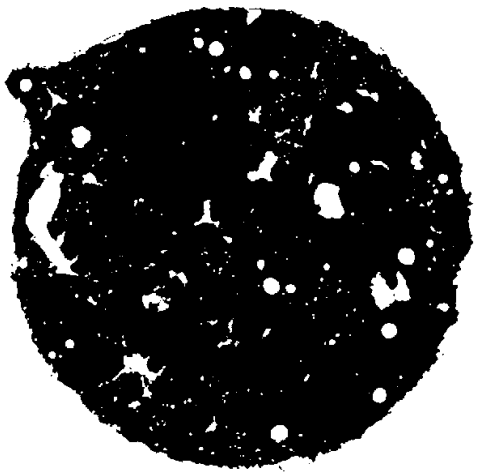
(c) Morphological Operations

Figure 4.4: Application of Morphological Operations on the 3D image.

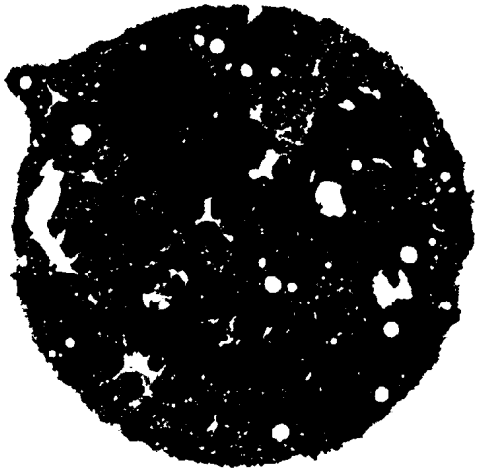
- Adding Small Pores: In some cases, very small pores are eliminated during the morphological opening operation. We add such small pores to the final image by setting the voxel values to one (background) in regions where the original sample data has intensity values less than one hundred. We also apply the same morphological operation as the step before to smoothen out the final result.



(a) Step



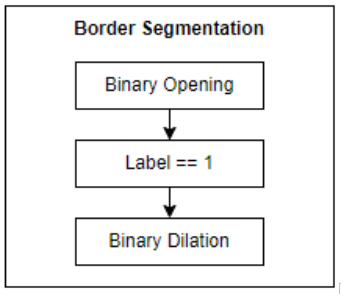
(b) Pores segmentation



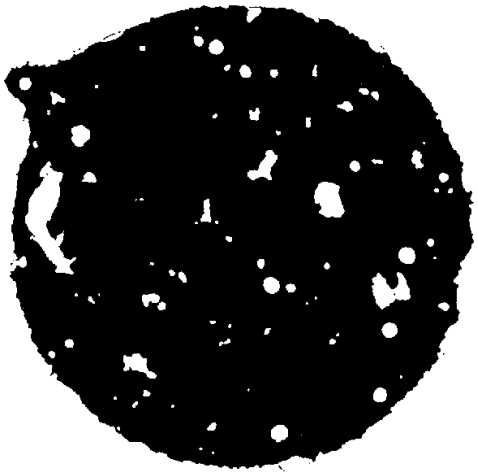
(c) Final pores segmentation

Figure 4.5: Adding the final touches to the pore segmentation.

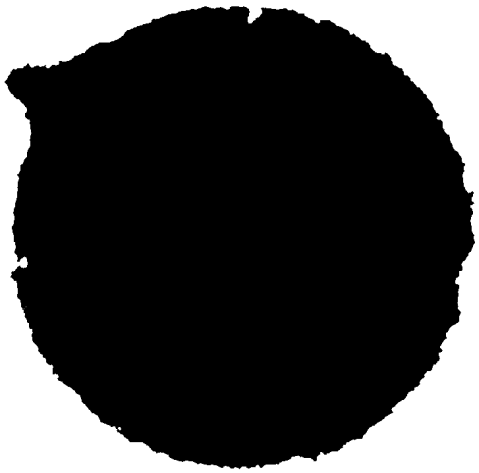
- Border Object: Finally, we apply a stronger binary opening operation once more to further isolate the pores ensuring they are well-defined and contiguous regions within the segmented image and are not attached to the border. We then label the binary image and isolate the border of our sample which is the biggest object assigned with the label number one.



(a) Step



(b) Border with pores



(c) Border

Figure 4.6: Border segmentation.

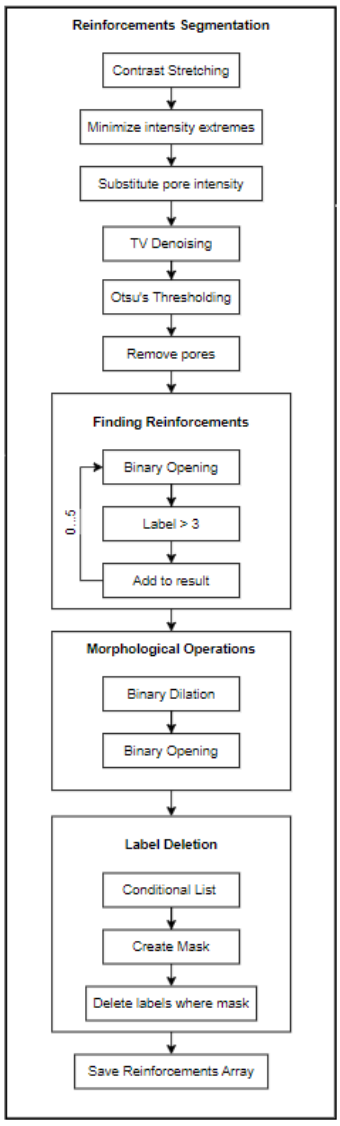
- Data Type Conversion: To facilitate compatibility with subsequent analysis and visualization steps, we convert the segmented pore image back to the appropriate data type (*np.byte*) and store it in files. To accomplish this we can either store them in a *NRRD* file using the nrrd package or as a *.pickle* file using the pickle package.

Pore identification is a critical component of our implementation, as it isolates the pores within the composite material, enabling us to quantitatively analyze their size, distribution, and other characteristics. The segmentation pipeline had good results for this part with the parameters used in record time. However for other samples there might be a need to change these parameters, namely the size of the structuring element used in the morphological operations. The resulting segmented image serves as a valuable

foundation for the subsequent stages of our thesis, where we delve into the identification and detailed characterization of the internal structure of the sample.

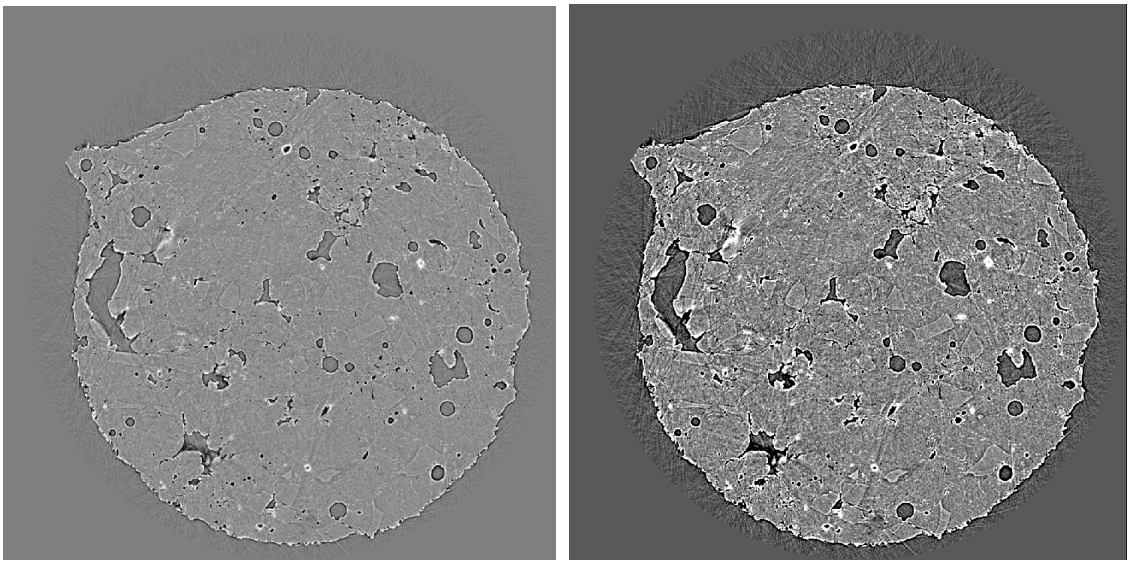
4.3 Reinforcement identification

In this step of our implementation, we focus on identifying and segmenting the reinforcement phases within the 3D composite material image. Here, we describe the steps taken to identify the reinforcements as well as the corresponding code snippet:



```
reinf = aux.contrast_stretch(image, 2, 98)
viewer.add_image(cp.asnumpy(reinf), name = 'contrast')

reinf[(reinf<100) | (reinf>200)] = cp.min(reinf)
reinf[pores==1] = 130
viewer.add_image(cp.asnumpy(reinf), name = 'min+avg')

reinf = aux.denoise_chunks(reinf, weight = 150, eps = 0.002, max_iter=50)
viewer.add_image(cp.asnumpy(reinf), name = 'denoise')

reinf = reinf >= filters.threshold_otsu(reinf)
reinf[pores==1] = 0
viewer.add_image(cp.asnumpy(reinf), name = 'thresh-pores')

result = cp.zeros_like(reinf)
for i in range(0,5):
    res = ndi.binary_opening(reinf, structure=morphology.ball(i))
    res,_ = ndi.label(res)
    res = (res > 3)
    viewer.add_image(cp.asnumpy(res), name = 'res')
    result += res
reinf = (result != 0)
viewer.add_image(cp.asnumpy(reinf), name = 'open')

reinf = ndi.binary_dilation(reinf, structure=morphology.ball(1))
reinf = ndi.binary_closing(reinf, structure=morphology.ball(1))
viewer.add_image(cp.asnumpy(reinf), name = 'morphops')

reinf, _ = ndi.label(reinf)
regions = measure.regionprops(reinf)

labels_to_delete = []
for region in regions:
    diameter = int(region.axis_major_length)
    if  diameter < 10 or int(region.area) < 2000 or diameter > 200:
        labels_to_delete.append(region.label)
labels_to_delete = cp.asarray(labels_to_delete)
mask = cp.isin(reinf, labels_to_delete)
reinf[mask] = 0
viewer.add_image(cp.asnumpy(reinf), name = 'elim')
```

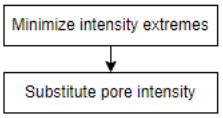
- Contrast Stretching: The initial step involves using the implemented contrast stretching function to set a new voxel intensity values range to the 3D image in *reinf* to enhance the visibility of the internal structure of the low-contrast sample.



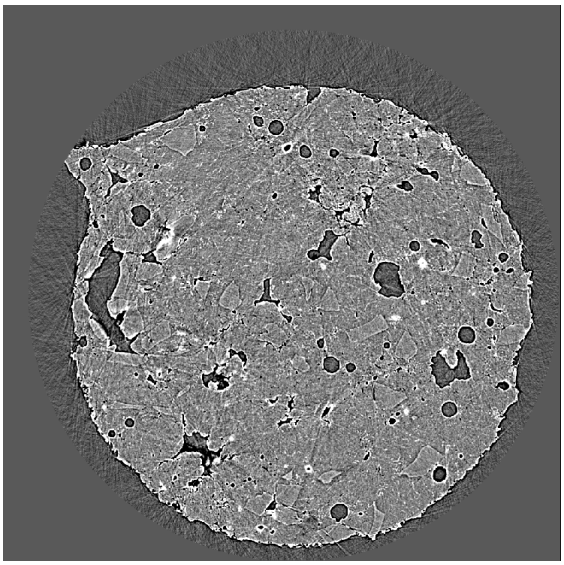
(a) Original 3D image (b) Contrast Stretching

Figure 4.7: Application of contrast stretching to the 3D image.

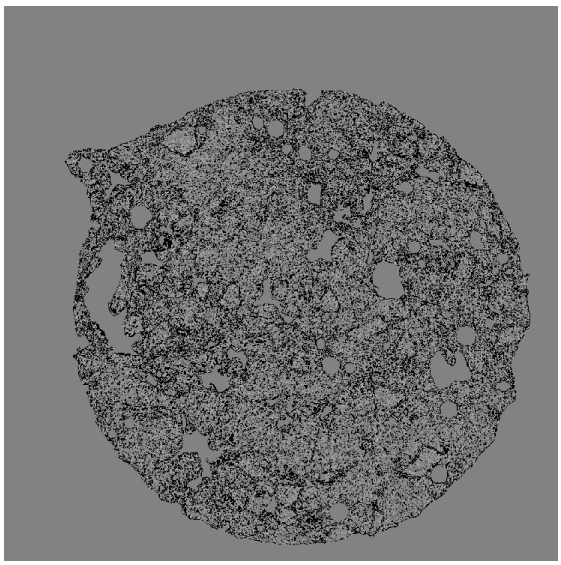
- Edge Enhancement: In this step, values in the *reinf* ndarray that fall below a threshold of one hundred or exceed two hundred are adjusted. Specifically, these values are set to the minimum value found within the *reinf* ndarray. This process helps to enhance the edges of the reinforcements for further processing. Also, we set the regions where there are pores to one hundred and thirty to match the average intensity of the reinforcements and base material.



(a) Step



(b) Contrast Stretching



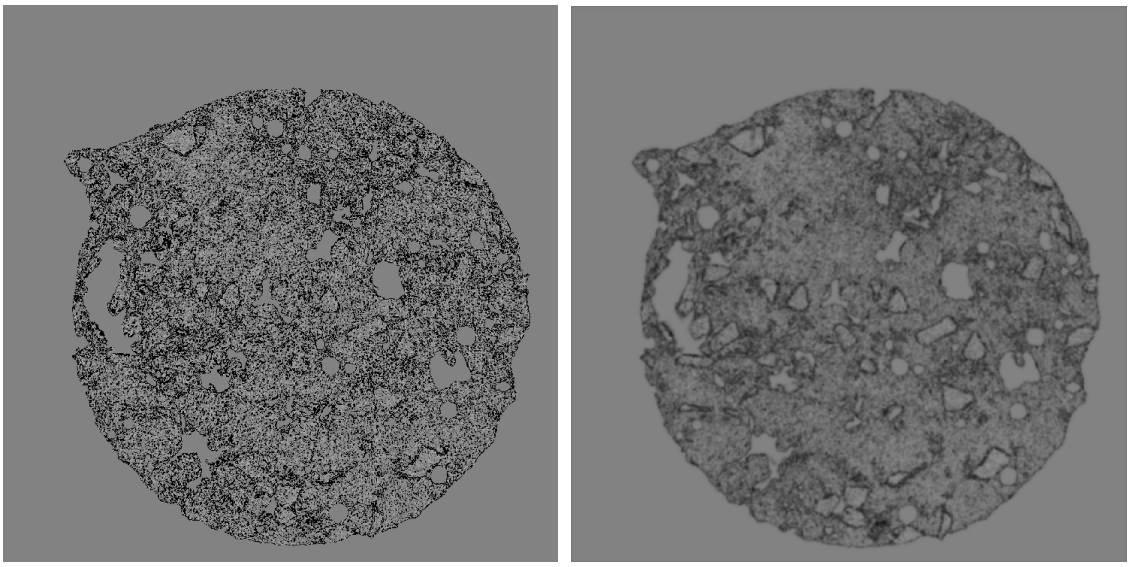
(c) Edge Enhancement

Figure 4.8: Edge enhancement and adjustments.

- TV Denoising: After experimentation with the *restoration.denoise_tv_chambolle()* cuCIM function's parameters, the *reinf* ndarray undergoes denoising. This step enhances image quality by reducing noise and preserving important features. This function is computationally expensive and creates arrays too big for the GPU. Therefore we created a *aux.denoise_chunks()* function:

```
def denoise_chunks(data, z_chunks = 10, weight = 10.0, eps = 2e-4, max_iter = 100):
    z_size, x_size, y_size = data.shape
    z_chunk_size = z_size // z_chunks
    data = cupy.asarray(data, dtype=np.float16)
    result = cupy.empty_like(data)

    for z_start in range(0, z_size, z_chunk_size):
        z_end = z_start + z_chunk_size if z_start + z_chunk_size <= z_size else
        ↪  z_size

        chunk = data[z_start:z_end, :, :]
        denoised_chunk = restoration.denoise_tv_chambolle(chunk, weight=weight, eps
        ↪  = eps, max_num_iter=max_iter)
        result[z_start:z_end, :, :] = denoised_chunk

    return cupy.asarray(result, dtype=np.ubyte)
```

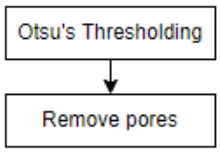
This function wraps the *restoration.denoise_tv_chambolle()* function and applies it to smaller sections of the 3D image.



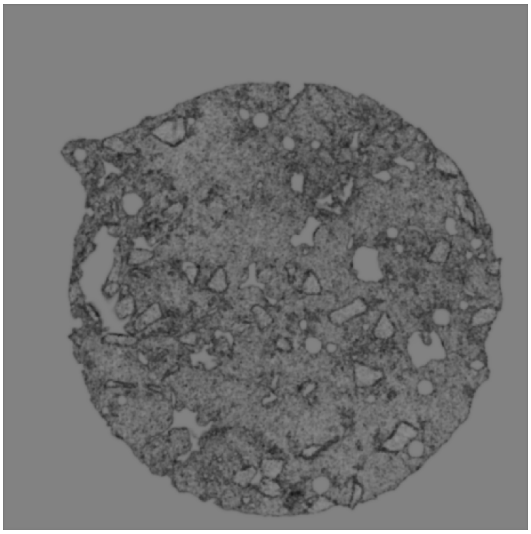
(a) Edge Enhancement (b) TV Denoising

Figure 4.9: Application of Chambolle's TV Denoising to the 3D image.

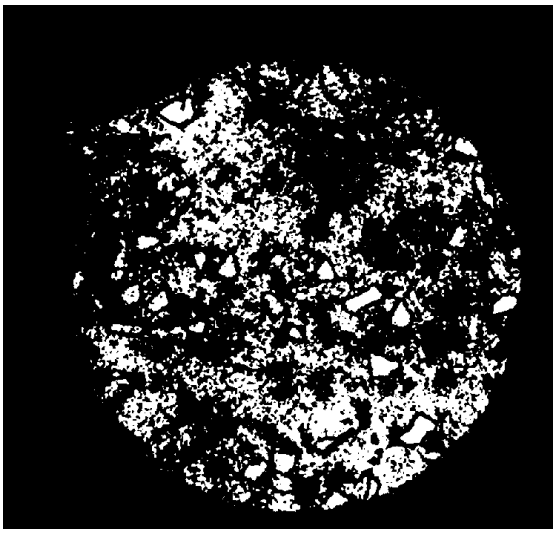
- Binary Thresholding and Pore Removal: The *reinf* ndarray is thresholded using Otsu's method (*filters.threshold_otsu()* from cuCIM) to create a binary image. This segmentation step identifies regions of interest in the image, in this case, the reinforcements and base material. Regions corresponding to the pores are set to zero as we are only interested in the reinforcements. Binary closing and opening operations are performed with spherical structuring elements of radii one and three, respectively, refining the binary mask by closing small holes present in the reinforcements and by separating them from each other and from the base material.



(a) Step



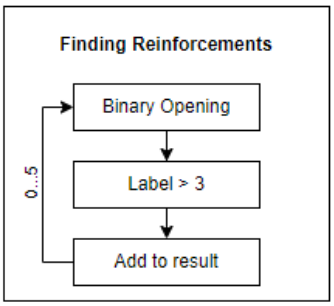
(b) TV Denoising



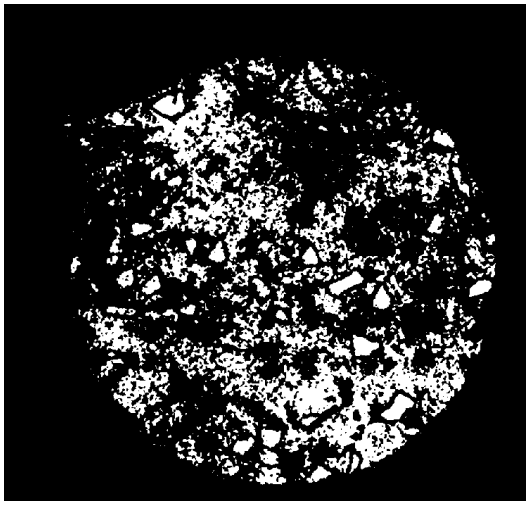
(c) Binary Thresholding

Figure 4.10: Application of Otsu's method to the 3D image.

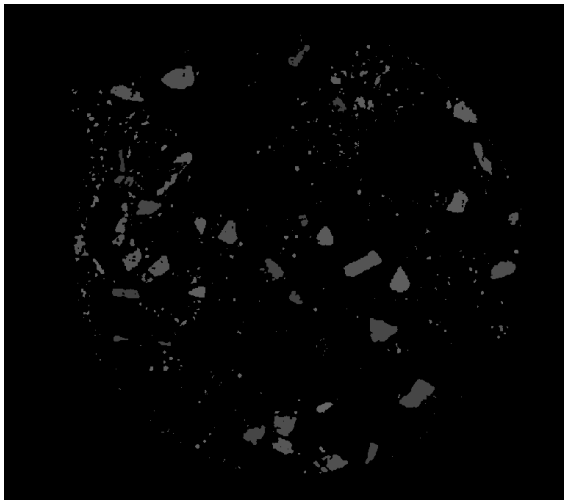
- Finding Reinforcements: At this time we have an *ndarray* that has some well defined reinforcements while other regions have reinforcements attached to the base material, as such, we need to separate them. We could use a single large sized structuring element for the morphological operation of opening to detach the reinforcements but not all of them would detach from the base material and some would even seize to exist. Hence, we iterate through different structure element sizes, discard the three first to guarantee that all reinforcements are correctly identified.



(a) Step



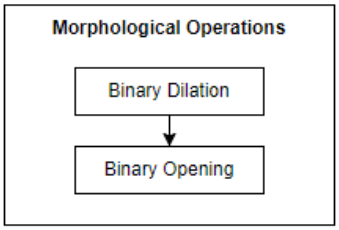
(b) Binary Thresholding



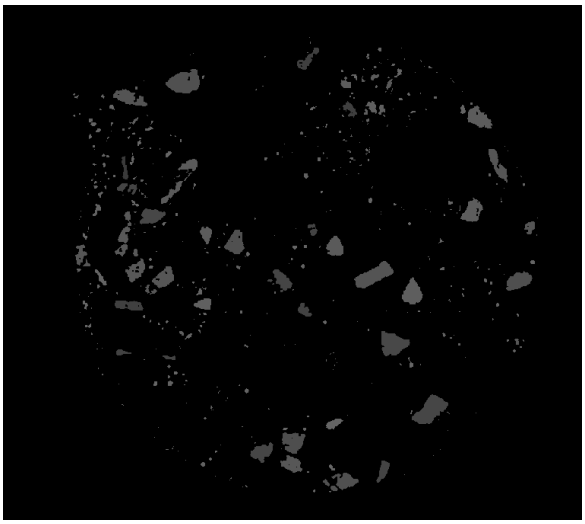
(c) Morphological Opening

Figure 4.11: Application of iterative morphological opening to the 3D binary image.

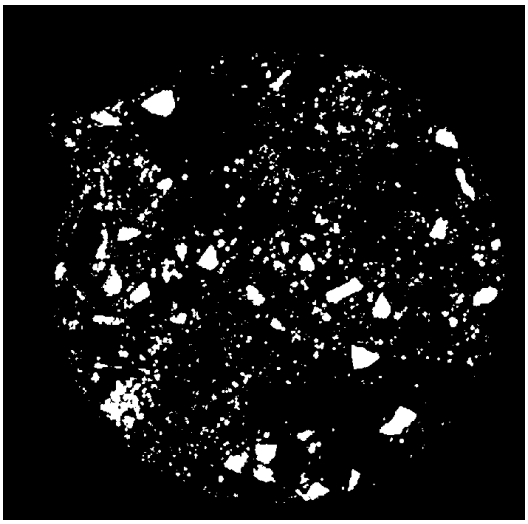
- Morphological Operations: Additional morphological operations, dilation and closing, are applied to further refine the resulting image to get the correct size for the reinforcements and to close any remaining holes.



(a) Step



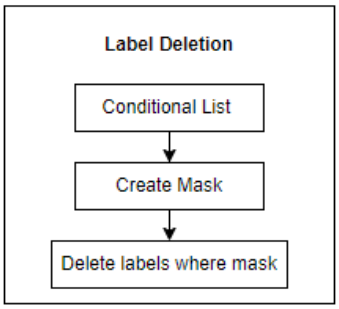
(b) Morphological Opening



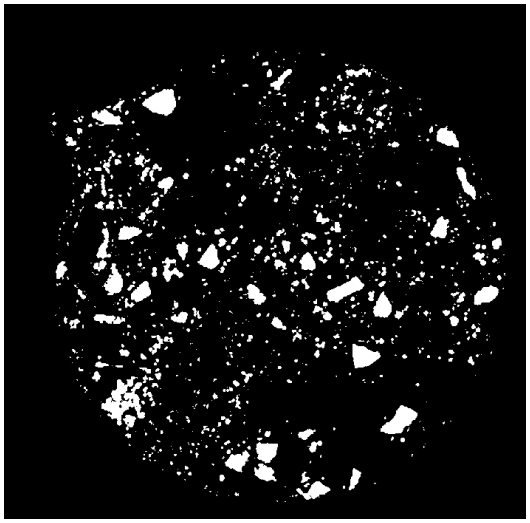
(c) Morphological Dilation and Closing

Figure 4.12: Further refinement of the 3D image with morphological operators.

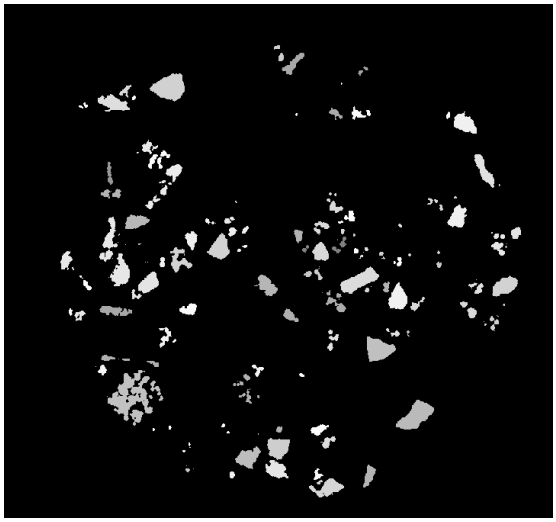
- Erasing Reinforcements: Since the image still contains some structures that might have detached from the base material when the morphological operations were applied or are remaining noise, they are not reinforcements and should be deleted. This step involves removing the labeled structures with volumes smaller than two thousand voxels and volumes with diameters smaller than ten voxels and bigger than two hundred voxels from the labeled image.



(a) Step



(b) Morphological Dilation and Closing



(c) Cleaned Image

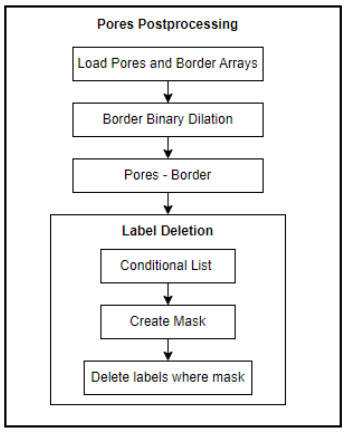
Figure 4.13: Erasure of *fake* reinforcements from the 3D image.

The resulting segmented image serves as a foundation for subsequent quantitative analyses, such as measuring the volume fraction, orientation, and distribution of reinforcement phases. As we can see, the final result can correctly identify the reinforcements but there are still some irregular structures that remain and that should be eliminated.

4.4 Post-processing

In this section, we delve into the final stages of our implementation, focusing on analysing the segmentation results of both pores and reinforcements and calculating the metrics for the next chapter.

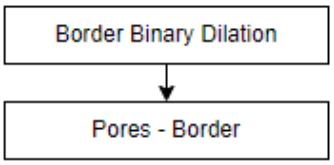
To address the pores we created a simple classification program where we divide them into spherical and irregular, corresponding to the types in section 3.2.3, respectively (see code below).



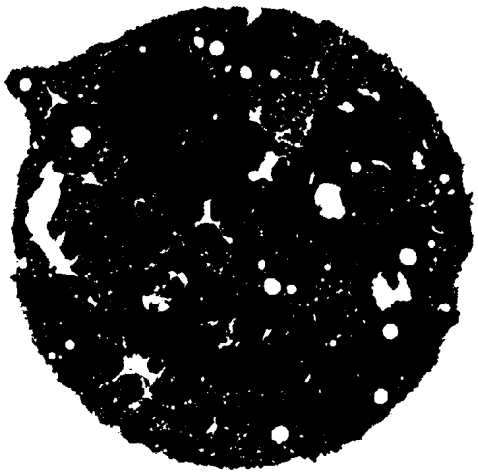

```
pores = aux.load_pickle("pores")
border = aux.load_pickle("border")
pores = cp.asarray(pores, dtype=np.byte)
border = cp.asarray(border, dtype=np.byte)

border = ndi.binary_dilation(border, structure=morphology.ball(5))
pores = pores - border
pores[pores == -1] = 0
viewer.add_image(cp.asnumpy(pores), name = 'onlypores')

pore_labels = measure.label(pores)
regions = measure.regionprops(pore_labels)

labels_to_delete = [prop.label for prop in regions if prop.extent < 0.3]
labels_to_delete = cp.asarray(labels_to_delete)
mask = cp.isin(pore_labels, labels_to_delete)
pore_labels[mask] = 0
viewer.add_labels(cp.asnumpy(mask), name = 'irregular')
viewer.add_labels(cp.asnumpy(pore_labels), name = 'spherical')
```

- Loading Data: Initially, the pores and border ndarrays are loaded from external files using the *aux.load_pickle()* function and then are converted into CuPy arrays with an unsigned byte data type (*np.byte*) and sent to the GPU.

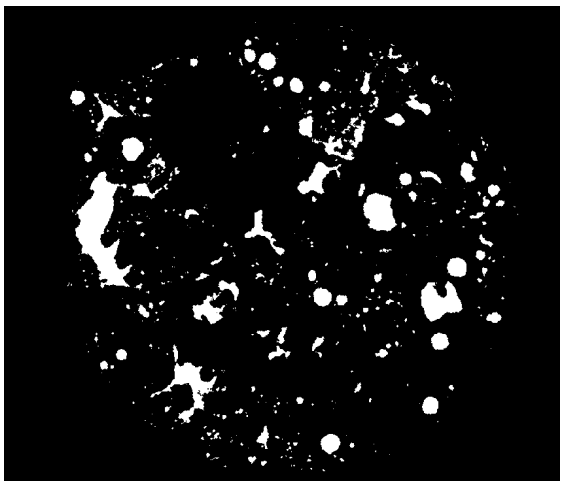
- Border Elimination: The *pores* ndarray contains both the pores and the border. Hence, we need to eliminate the border without also erasing pores that are connected to said border. A binary dilation operation is performed on the border array using a spherical structuring element with a radius of five expanding the white regions of the border. The pores array is subtracted by the dilated border array. This operation is used to separate regions of interest (pores) from the border regions, resulting in a binary image with isolated pore structures. Due to the subtraction, some of the voxels between the border and the pores are assigned the value minus one due to the dilation operation not resulting into equal borders, and as such are also eliminated.



(a) Step



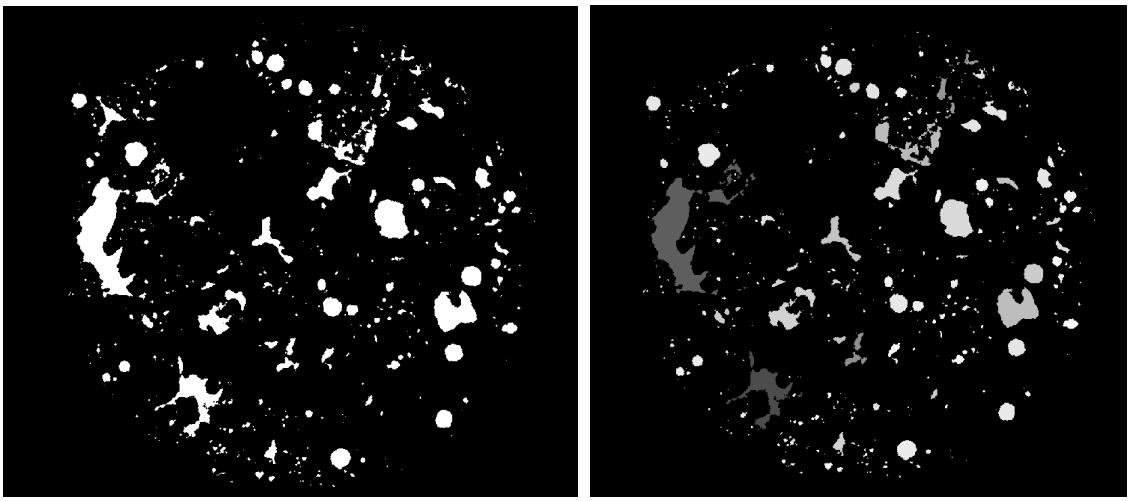
(b) Pores and Border



(c) Only Pores

Figure 4.14: Subtraction of the border from the 3D image.

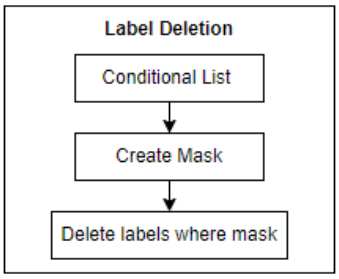
- Labeling Pores and Region Properties Calculation: The pores array is labeled using the *measure.label()* function from cuCIM. This operation assigns a unique label to each connected component in the binary image, effectively labeling individual pores. Using *measure.regionprops()* from cuCIM, region properties (e.g., area, centroid, extent) are computed for each labeled region. This step is useful for characterizing and analyzing the identified pores.



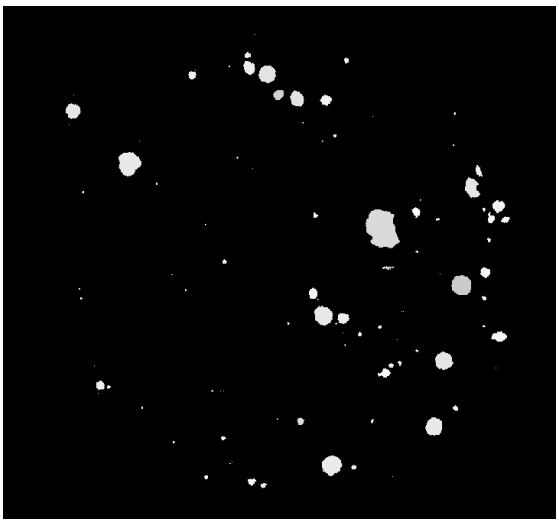
(a) Pores (b) Labeled Pores

Figure 4.15: Application of a labelling algorithm to the 3D binary image.

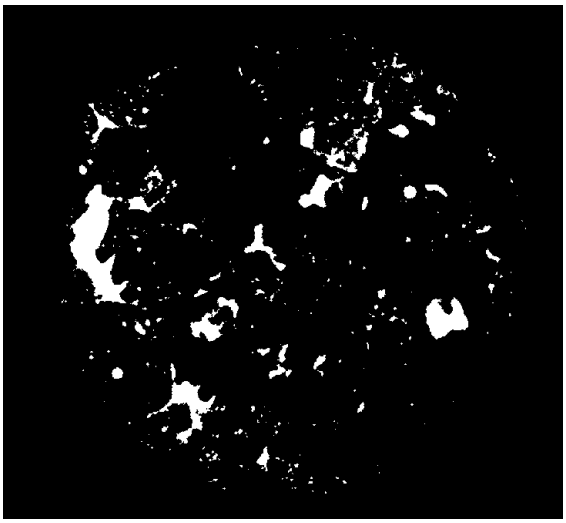
- Label Deletion Based on Extent: Labels of regions that have an extent (ratio of pixels in the region to pixels in the total bounding box) less than three decimals are identified and collected in the *labels_to_delete* list. The *labels_to_delete* list is converted into a CuPy array for efficient computation and a binary mask is created using *cp.isin()* to filter out labels that should be deleted. This mask identifies regions in the *pore_labels* array that match the labels in *labels_to_delete*. The identified labels are deleted from the *pore_labels* array by setting them to zero. This step effectively removes undesired labels, corresponding to aspherical pore regions, from the labeled image. Hence, the *pore_labels* ndarray contains the spherical pores while the *mask* ndarray contains the irregular pores.



(a) Step



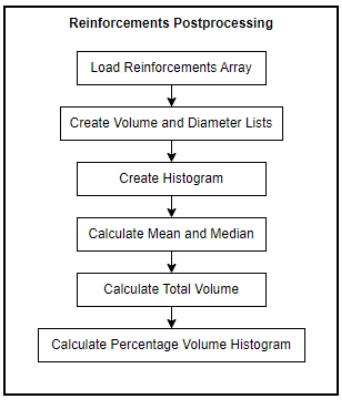
(b) Spherical



(c) Irregular

Figure 4.16: Separation of the pores ndarray into 2 classes.

For the next chapter, there is a need to calculate the granulometric distribution of the reinforcements in the 3D image. Below, we outline the steps involved:




```
regions = measure.regionprops(labeled_image)
diameters = [int(region.axis_major_length) for region in regions]
volumes = [int(region.area_filled) for region in regions]

bin_edges = np.linspace(min(diameters), max(diameters), num=40)
hist, _ = np.histogram(diameters, bins=bin_edges, weights=volumes)

mean = np.mean(diameters)
median = np.median(diameters)

total_volume = np.sum(volumes)
percentage_volume = (hist / total_volume) * 100

plt.plot(bin_edges[:-1], percentage_volume)
plt.xlabel("Diameter (µm)")
plt.ylabel("Percentage of Volume (%)")
plt.title("Volumetric Distribution vs. Diameter (µm)")
plt.grid(False)
plt.show()
```

- Volume and Diameter Calculation: We begin by calculating properties of labeled regions within the segmented image using *measure.regionprops()*. This function can calculate many properties, namely the diameter (axis_major_length) and the volume (area_filled) of each region. We iterate through the calculated region properties extracting the diameters and volumes.

- Histogram Creation: Using the collected diameters and volumes data, we create a histogram to represent the volumetric distribution of reinforcement sizes. We define bin edges for the histogram using *np.linspace()* and then calculate the histogram itself using *np.histogram()*.

- Statistical Analysis: We calculate the mean and median diameter of the reinforcements. These measures offer a comprehensive view of the particle size distribution and central tendencies that will be talked about in the next chapter.
- Percentage Volume Calculation: To better understand the relative volume occupied by particles of different sizes, we calculate the percentage volume for each bin in the histogram. This percentage is derived by dividing the volume within each bin by the total volume of reinforcements and multiplying it by one hundred.
- Visualization: Finally, we plot the volumetric distribution against the diameters (in micrometers) using a line plot. The x-axis represents the diameter bins, and the y-axis represents the percentage of volume occupied by particles within each bin. This visualization provides a clear representation of the particle size distribution.

We have also calculated the total percentage of reinforcements in the sample:

```
1 def calc_total_percentage_reinf(image, pore_labels, reinf_labels):
2     total_size = image.shape[0]*image.shape[1]*image.shape[2]
3     return (total_size - total_area(pore_labels))/total_area(reinf_labels)
```

The ratio of the difference between the total area of the image and the total area of the pores to the total area of the segmented regions in *reinf* yields the total percentage of reinforcements of the composite material.

5

Evaluation and discussion of the results

In this chapter, this thesis presents the results obtained from an innovative segmentation pipeline applied to a low-contrast composite material sample. These results will allow researchers to better characterize and analyse the underlying structure of these composite materials.

In the following sections, we delve into the specifics of our segmentation pipeline, discuss the methodologies employed, the processing time of the pipeline and present the outcomes in terms of segmented material phases and structures. We highlight the implications of these results in the field of materials science, emphasizing how they contribute to our understanding of materials at the microscopic level and, consequently, drive advancements in this domain.

5.1 Performance Evaluation

This section presents the execution times of the segmentation pipeline offering a detailed examination of its efficiency and effectiveness in segmenting a low-contrast composite material sample with a volume of 432x1024x1024 (Z,Y,X axis) voxels, particularly in the context of GPU (Graphics Processing Unit) versus CPU (Central Processing Unit) utilization. By comparing the execution times on both hardware platforms, we aim to assess the advantages and drawbacks of each approach, shedding light on the potential performance gains achieved by harnessing the computational power of GPUs. This section will not only validate the pipeline's functionality but also serve as a foundation for making informed decisions regarding hardware selection and optimization to maximize its overall performance.

The program was run in a Ubuntu 20.04.6 LTS (64-bit) environment in a machine with the following specifications:

CPU	AMD® Ryzen 5 3600x 6-core processor × 12
GPU	NVIDIA Corporation TU106 [GeForce RTX 2060 Rev. A] 6GB
RAM	16GB

Table 5.1: Machine Specifications.

Below we have a table with the execution times of the processing pipeline. It is divided into two: pore and reinforcement, with application of segmentation and classification. Notice that the segmentation is further divided into thresholding, labeling and cleaning.

Operations		Execution Time (s)	
		Pores	Reinforcements
Segmentation	Thresholding	1.14	7.55
	Labeling	0.81	4.29
	Cleaning	0.43	14.02
Classification		7.64	3.57
Total		9.13	30.38

Table 5.2: Execution times (s).

It is essential to acknowledge the significant impact of processing time in our implementation, particularly in the context of CPU and GPU utilization. The power of modern GPUs in accelerating image processing tasks cannot be overstated. Leveraging GPU parallelism, we observed substantial reductions in computation time compared to a CPU-only approach, thus allowing us to handle larger datasets.

As we can see in table below, if we compare the execution time of the pore segmentation we see that there is a x50 times improvement of the GPU compared to the CPU in the thresholding step, a x5 times improvement in the labelling step and a x240 times improvements in the cleaning step.

Pores	Execution Time (s)	
Segmentation	CPU	GPU
Thresholding	56.12	1.14
Labeling	4.42	0.81
Cleaning	103.32	0.43
Total	163.95	2.38

Table 5.3: Comparison of execution times between the CPU and the GPU-only approach.

Nevertheless, we also acknowledge that GPU-based implementations may require specialized hardware and expertise. Therefore, in the implementation of our solution we considered both CPU and GPU implementations and used packages that allowed for interchangeability. This inclusive approach ensures that any researcher can use this segmentation pipeline.

5.2 Segmentation Results Assessment

This section provides valuable metrics to substantiate the results obtained, where the quality and accuracy of the segmentation process are assessed. The volumetric distribution of reinforcements within the segmented image is analyzed, offering valuable insights into the spatial characteristics of these critical components. An experts opinion is also considered, providing a qualitative dimension to the evaluation process.

5.2.1 Reinforcement Segmentation Results

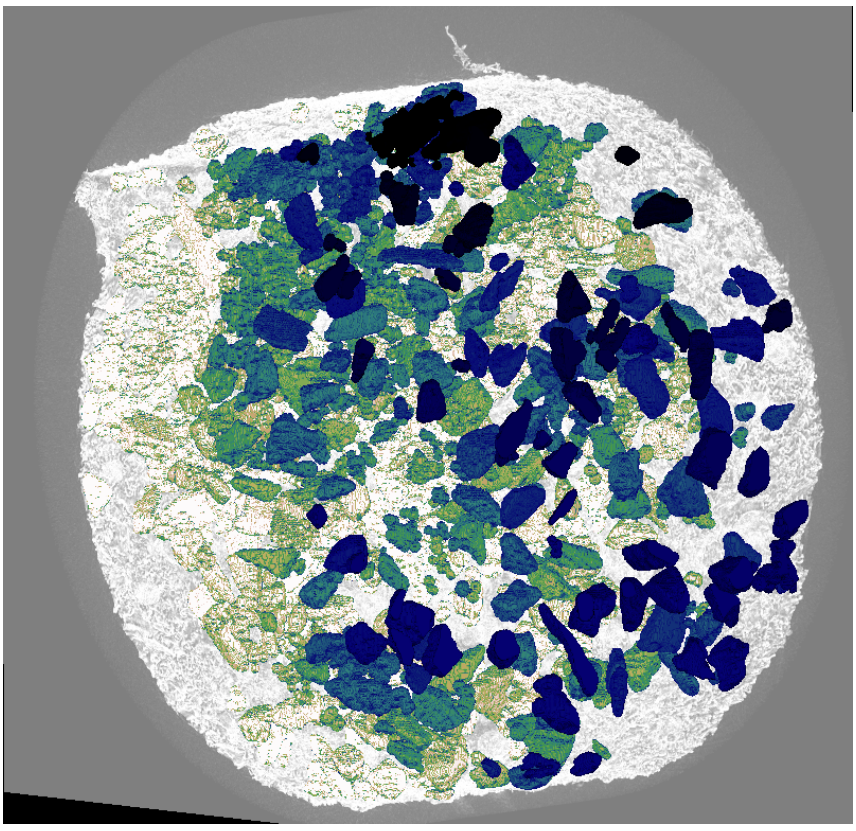


Figure 5.1: Result of the reinforcement segmentation pipeline.

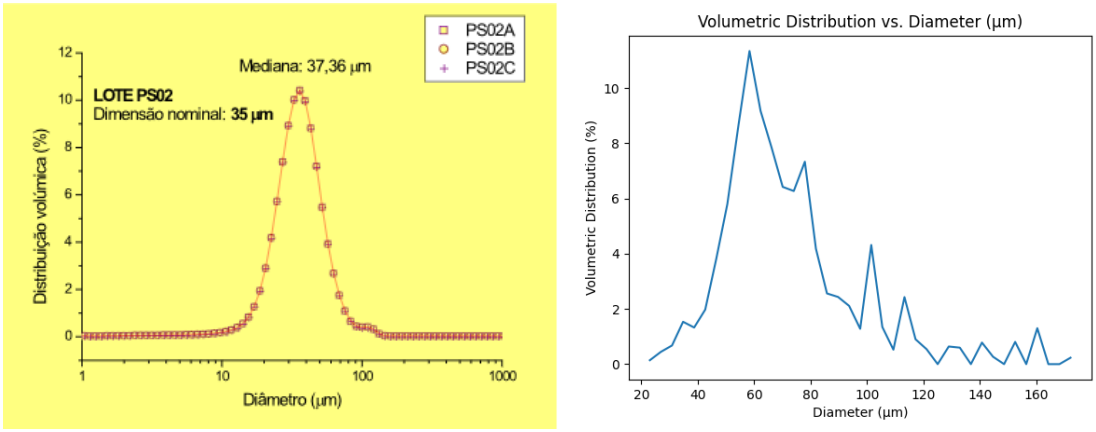
Figure 5.1 shows a 3D representation of the sample and the reinforcements within. We can see that overall, the results look promising being that it correctly segments most of the reinforcements in the sample. We can also see groups of small spherical objects

through the sample which are considered *fake* reinforcements which are small pieces of base material that .

It is to note that all reinforcements can be characterized by several metrics which are calculated and written in a Excel file for further analysis and study by researchers later on.

5.2.2 Validation by total percentage and volumetric distribution of reinforcements

Among the validation criteria, assessing the total percentage and volumetric distribution of reinforcements within the sample assists in confirming the precision and reliability of the analysis.



(a) Ground truth histogram

(b) Obtained histogram

Figure 5.2: Granulometric analysis comparison between the ground truth data and the result obtained.

Validation by both total percentage and volumetric distribution involves comparing the pipeline's results to ground truth data. The only ground truth data available was obtained from Professor Velhinho, shown in Fig.5.2a. We analysed the sample in the previous chapter, section 4.4, and obtained the particle-size distribution, shown in Fig.5.2b.

The total percentage of reinforcements in the sample has a percentage error of 4.8% to the described 10% in section 3.2.2. The validation metrics for the volumetric distribution have a percentage error of 8.11% for the median (34.0) and 22.09% for the mean (42.73) which falls within expectation, reinforcing the robustness of the segmentation pipeline. These small percentage errors are largely due to the presence of *fake* reinforcements. These are normally located in between pores and reinforcements. If we look at the percentage distribution of the reinforcements, we can see that there is a general increase of the percentage of volume of reinforcements which indicates that there are more reinforcements present in the results than expected. Notably, there is some large spikes after the one hundred micrometer diameter mark, showing that there are big structures within the segmentation result that are either *fake* reinforcements or that there are some fused

structures (morphological dilation or closing operations can attach structures together if they are very close to each other).

These values closely match, serving as compelling evidence of the segmentation pipeline's accuracy.

5.2.3 Experts opinion

Professor Alexandre Velhinho, who is an expert in materials science and a professor of the Material Science Department at NOVA School of Science and Technology, offered his perspective on the results obtained from the segmentation pipeline applied to the low-contrast composite material sample. His extensive experience in the field and deep knowledge of the sample made his assessment particularly insightful.

The correct segmentation of low-contrast samples has always posed a significant challenge. These materials often exhibit subtle variations in composition and structure, which can be exceedingly difficult to capture and quantify accurately through traditional imaging methods that were used in the department. The results presented in this study, obtained through an innovative segmentation pipeline show a promising technique to achieve accurate segmentation of low-contrast samples. There is still room for improvement, seeing that artefacts and *fake* reinforcements may persist in the final result and that some true reinforcements may not be accurately segmented. However, overall this method correctly segments most of the sample with good accuracy.

6

Conclusion and future work

In this thesis, we have successfully applied a new image segmentation technique for low-contrast images, which resulted in satisfactory outcomes and with reasonable execution times. Our chosen method allowed for the precise delineation of structures within the images, facilitating their characterization and enabling in-depth analysis.

Furthermore, it is important to emphasize that in applications where we have large datasets, harnessing the computational power of Graphics Processing Units (GPUs) is not merely advantageous but imperative for achieving low execution times. GPU utilization is becoming increasingly indispensable in these scenarios, allowing for rapid and efficient processing of large datasets and complex image segmentation pipelines.

While the results obtained in this thesis are encouraging as discussed in the previous chapter, there remain other segmentation methods that could be used in the future for further refinements in image segmentation. Investigating the incorporation of deep learning models, such as Convolutional Neural Networks (CNNs), for segmentation purposes could lead to more robust and adaptable methods. These techniques often excel in handling complex and intricate structures within images. Training machine learning models for segmentation tasks can enable the automatic adaptation of segmentation strategies to specific datasets or applications. This adaptive nature can significantly improve segmentation accuracy and generalizability.

In conclusion, this thesis has established a robust foundation by demonstrating the effectiveness of our segmentation technique for accurate object delineation, characterization, and analysis.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] J. C. Elliott and S. D. Dover. "X-ray microtomography". In: *Journal of Microscopy* 126.2 (1982), pp. 211–213. DOI: https://doi.org/10.1111/j.1365-2818.1982.tb00376.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2818.1982.tb00376.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2818.1982.tb00376.x (cit. on pp. 1, 6).

[3] Y. Wang and J. Miller. "Current developments and applications of micro-CT for the 3D analysis of multiphase mineral systems in geometallurgy". In: *Earth-Science Reviews* 211 (2020), p. 103406. ISSN: 0012-8252. DOI: https://doi.org/10.1016/j.earscirev.2020.103406. URL: https://www.sciencedirect.com/science/article/pii/S0012825220304529 (cit. on pp. 1, 26).

[4] I. Varfolomeev, I. Yakimchuk, and I. Safonov. "An Application of Deep Neural Networks for Segmentation of Microtomographic Images of Rock Samples". In: *Computers* 8.4 (2019). ISSN: 2073-431X. DOI: 10.3390/computers8040072. URL: https://www.mdpi.com/2073-431X/8/4/72 (cit. on pp. 2, 14).

[5] *Paleo Imaging Workshop Day 2: Data Processing in Image/Beginning Visualization with Avizo*. 2014. URL: http://idigbio.adobeconnect.com/p3lfifgah5a/ (cit. on p. 2).

[6] *VGSTUDIO Max*. URL: https://www.volumegraphics.com/en/products/vgsm.html (cit. on p. 2).

[7] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics*. Fourth. Kitware, Inc., 2006 (cit. on p. 2).

[8] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000) (cit. on pp. 2, 16, 18).

[9] NVIDIA, P. Vingelmann, and F. H. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit (cit. on pp. 3, 4).

[10] Khronos OpenCL Working Group. *The OpenCL Specification, Version 1.1*. Ed. by A. Munshi. 2011. URL: https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (cit. on pp. 3, 4).

[11] S. K. Lam, A. Pitrou, and S. Seibert. "Numba: A llvm-based python jit compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6 (cit. on pp. 3, 4).

[12] C. Lattner and V. Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation". In: *CGO*. San Jose, CA, USA, 2004, pp. 75–88 (cit. on p. 3).

[13] C. R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020-09), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 3).

[14] J. M. M. Ribeiro. *3D GPU-enabled Connected-Component Labeling Algorithms with Numba*. 2021-10 (cit. on pp. 4, 9, 10, 15, 17, 18, 32).

[15] R. Okuta et al. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf (cit. on pp. 4, 19).

[16] A. Klöckner et al. "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation". In: *Parallel Computing* 38.3 (2012-03), pp. 157–174. ISSN: 0167-8191. DOI: 10.1016/j.parco.2011.09.001. URL: http://www.sciencedirect.com/science/article/pii/S0167819111001281 (visited on 2012-02-26) (cit. on p. 4).

[17] L. Vásárhelyi et al. "Microcomputed tomography–based characterization of advanced materials: a review". In: *Materials Today Advances* 8 (2020), p. 100084. ISSN: 2590-0498. DOI: https://doi.org/10.1016/j.mtadv.2020.100084. URL: https://www.sciencedirect.com/science/article/pii/S259004982030031X (cit. on p. 7).

[18] U. Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Clifton Park, NY, USA: Kitware, Inc., 2015. ISBN: 1930934300. DOI: 10.5555/2789330 (cit. on pp. 7, 13, 19).

[19] G. Kindlmann et al. *NRRD*. URL: https://teem.sourceforge.net/nrrd/ (cit. on p. 8).

[20] F. Birra et al. "Tomographic image analysis of reinforcement distribution in composites using a flexible and material's specialist-friendly computational environment". In: *Materials Letters: X* 7 (2020), p. 100046. ISSN: 2590-1508. DOI: https://doi.org/10.1016/j.mlblux.2020.100046. URL: https://www.sciencedirect.com/science/article/pii/S2590150820300090 (cit. on pp. 8, 9).

[21] G. L. Vignoles. "Image segmentation for phase-contrast hard X-ray CMT of C/C composites". In: *Carbon* 39.2 (2001), pp. 167–173. DOI: 10.1016/S0008-6223(00)00103-2. URL: https://hal.science/hal-00327612 (cit. on p. 9).

[22] B. D. A. Brito. *Tomographic Image Processing Using Julia and GPU*. 2022-10 (cit. on pp. 9, 10).

[23] A. Velhinho et al. "A Problem-Solving Environment for reinforcement distribution characterization in composites using tomographic images". In: *Ciencia e Tecnologia dos Materiais* 24 (2012-01), pp. 149–152 (cit. on p. 11).

[24] P. Quaresma et al. "PSE for Analysis of 3D Tomographic Images in Materials Science". In: *Preprocedings of the 18th Conference on Computer Science and Intelligence Systems pp. 1107-1111* (2023) (cit. on pp. 10, 22, 23).

[25] F. Shih. *Image Processing and Pattern Recognition: Fundamentals and Techniques, Wiley-IEEE Press*. 2010-12. ISBN: 978-0-470-40461-4. DOI: 10.1002/9780470590416 (cit. on p. 11).

[26] M. Sezgin, B. Sankur, and B. I. "Survey Over Image Thresholding Techniques". In: *J Electronic Imaging* 13 (2004-07) (cit. on pp. 12, 13).

[27] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076 (cit. on pp. 12, 30).

[28] M. Zeng et al. "Improving histogram-based image contrast enhancement using gray-level information histogram with application to X-ray images". In: *Optik* 123.6 (2012), pp. 511–520. ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2011.05.017. URL: https://www.sciencedirect.com/science/article/pii/S003040261100249X (cit. on p. 13).

[29] J. Song et al. "A two-stage adaptive thresholding segmentation for noisy low-contrast images". In: *Ecological Informatics* 69 (2022), p. 101632. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2022.101632. URL: https://www.sciencedirect.com/science/article/pii/S1574954122000814 (cit. on p. 13).

[30] Z. Peter et al. "A constrained region growing approach based on watershed for the segmentation of low contrast structures in bone micro-CT images". In: *Pattern Recognition* 41.7 (2008), pp. 2358–2368. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2007.12.011. URL: https://www.sciencedirect.com/science/article/pii/S0031320307005390 (cit. on p. 13).

[31] V. Ortalan et al. "Application of image processing to STEM tomography of low-contrast materials". In: *Ultramicroscopy* 110.1 (2009), pp. 67–81. ISSN: 0304-3991. DOI: https://doi.org/10.1016/j.ultramic.2009.09.007. URL: https://www.sciencedirect.com/science/article/pii/S0304399109002034 (cit. on p. 13).

[32] Z. Al-Ameen et al. "An innovative technique for contrast enhancement of computed tomography images using normalized gamma-corrected contrast-limited adaptive histogram equalization". In: *EURASIP Journal on Advances in Signal Processing* 2015 (2015-04), pp. 1–12. DOI: 10.1186/s13634-015-0214-1 (cit. on p. 13).

[33] P. Bhattad, C. S. Willson, and K. E. Thompson. "Segmentation of Low-contrast Three-phase X-ray Computed Tomography Images of Porous Media". In: *Advances in Computed Tomography for Geomaterials*. John Wiley Sons, Ltd, 2010, pp. 254–261. ISBN: 9781118557723. DOI: https://doi.org/10.1002/9781118557723.ch30. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118557723.ch30. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118557723.ch30 (cit. on p. 13).

[34] J. CANNY. "A Computational Approach to Edge Detection". In: *Readings in Computer Vision*. Ed. by M. A. Fischler and O. Firschein. San Francisco (CA): Morgan Kaufmann, 1987, pp. 184–203. ISBN: 978-0-08-051581-6. DOI: https://doi.org/10.1016/B978-0-08-051581-6.50024-6. URL: https://www.sciencedirect.com/science/article/pii/B9780080515816500246 (cit. on p. 14).

[35] L. He et al. "Fast connected-component labeling". In: *Pattern Recognition* 42.9 (2009), pp. 1977–1987. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2008.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0031320308004573 (cit. on p. 14).

[36] L. He et al. "The connected-component labeling problem: A review of state-of-the-art algorithms". In: *Pattern Recognition* 70 (2017), pp. 25–43. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.04.018. URL: https://www.sciencedirect.com/science/article/pii/S0031320317301693 (cit. on pp. 14, 15).

[37] X. Zhao et al. "An Efficient Connected-Component Labeling Algorithm for 3-D Binary Images". In: *IEEE Open Journal of the Computer Society* 4 (2023), pp. 1–12. DOI: 10.1109/OJCS.2022.3233088 (cit. on pp. 16, 18).

[38] N. Maurice et al. "LSL3D: a run-based Connected Component Labeling algorithm for 3D volumes". In: *Binary is the new Black and White workshop @ IEEE ICIAP 2022*. Lecce, Italy, 2022-05. URL: https://hal.science/hal-03689455 (cit. on p. 16).

[39] P. Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on pp. 16, 19).

[40] S. Van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (2014), e453 (cit. on pp. 16, 19).

[41] N. Sofroniew et al. *napari/napari: 0.4.13rc0*. Version v0.4.13rc0. 2022-01. DOI: 10.5281/zenodo.5848842. URL: https://doi.org/10.5281/zenodo.5848842 (cit. on pp. 17, 19).

[42] D. Clark and C. Badea. "Micro-CT of rodents: State-of-the-art and future perspectives". In: *Physica Medica* 30.6 (2014), pp. 619–634. ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.ejmp.2014.05.011. URL: https://www.sciencedirect.com/science/article/pii/S1120179714001008 (cit. on p. 18).

[43] L. Pan, L. Gu, and J. Xu. "Implementation of medical image segmentation in CUDA". In: *2008 International Conference on Information Technology and Applications in Biomedicine*. 2008, pp. 82–85. DOI: 10.1109/ITAB.2008.4570542 (cit. on p. 18).

[44] A. Prahara et al. "Parallel Approach of Adaptive Image Thresholding Algorithm on GPU". In: *Knowledge Engineering and Data Science* 4.2 (2022), pp. 69–84. ISSN: 2597-4637. DOI: 10.17977/um018v4i22021p69-84. URL: http://journal2.um.ac.id/index.php/keds/article/view/25980 (cit. on p. 18).

[45] M. H. Najafi et al. "GPU-Accelerated Nick Local Image Thresholding Algorithm". In: *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. 2015, pp. 576–584. DOI: 10.1109/ICPADS.2015.78 (cit. on p. 18).

[46] F. Bolelli, S. Allegretti, and C. Grana. "Connected Components Labeling on Bitonal Images". In: *Image Analysis and Processing – ICIAP 2022*. Ed. by S. Sclaroff et al. Cham: Springer International Publishing, 2022, pp. 347–357. ISBN: 978-3-031-06430-2. DOI: 10.1007/978-3-031-06430-2_29 (cit. on p. 18).

[47] *Welcome to CUCIM's documentation!* URL: https://docs.rapids.ai/api/cucim/stable/ (cit. on p. 19).

[48] R. Haase et al. *clij/clij2: 2.2.0.1*. Version 2.2.0.1. 2020-10. DOI: 10.5281/zenodo.4134962. URL: https://doi.org/10.5281/zenodo.4134962 (cit. on p. 19).

[49] J. Schindelin et al. "Fiji: an open-source platform for biological-image analysis". In: *Nat Meth* 9.7 (2012-07), pp. 676–682. ISSN: 15487091. URL: http://dx.doi.org/10.1038/nmeth.2019 (cit. on p. 19).

[50] T. Barbu. "Variational Image Denoising Approach with Diffusion Porous Media Flow". In: *Abstract and Applied Analysis* 9 (2013), pp. 25–30. DOI: 10.1155/2013/856876 (cit. on p. 25).

[51] F. E. Boas, D. Fleischmann, et al. "CT artifacts: causes and reduction techniques". In: *Imaging Med* 4.2 (2012), pp. 229–240 (cit. on p. 26).

[52] A. Jain. *Fundamentals of Digital Image Processing*. Vol. 7. Prentice-Hall information and system sciences series. Prentice Hall, 1989, p. 258. ISBN: 9780133361650. URL: https://books.google.pt/books?id=mQNSAAAAMAAJ (cit. on p. 27).

[53] K. Zuiderveld. "Contrast limited adaptive histogram equalization". In: *Graphics gems* (1994), pp. 474–485 (cit. on p. 28).

[54] M. Celenk et al. "Upper airway detection and visualization from cone beam image slices". In: *Journal of X-ray science and technology* 18 (2010-01), pp. 121–35. DOI: 10.3233/XST-2010-0248 (cit. on p. 29).

[55] P. Soille. "Erosion and Dilation". In: *Morphological Image Analysis: Principles and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–103. ISBN: 978-3-662-05088-0. DOI: 10.1007/978-3-662-05088-0_3. URL: https://doi.org/10.1007/978-3-662-05088-0_3 (cit. on p. 30).

[56] V. Mardiris and V. Chatzis. "A Configurable Design for Morphological Erosion and Dilation Operations in Image Processing using Quantum-dot Cellular Automata". In: *Journal of Engineering Science and Technology Review* 9 (2016-05), pp. 25–30. DOI: 10.25103/jestr.092.05 (cit. on pp. 30, 31).

[57] A. Chambolle. "An algorithm for total variation minimization and applications". In: *Journal of Mathematical Imaging and Vision* 20 (2004), pp. 89–97. DOI: 10.1155/2013/856876 (cit. on p. 31).