Colon Segmentation Pipeline for Abdominal CT Scans

Pedro Freire

Contents

1 Introduction

Colorectal cancer and other gastrointestinal disorders require accurate and efficient imaging techniques for diagnosis and treatment planning. Abdominal CT scans are commonly used for this purpose, often requiring dietary preparation with contrast ingestion and colon inflation to enhance visibility. Traditional methods for virtual colonoscopy involve taking two CT scans in different patient positions (supine and prone) to ensure that all regions of the colon are visible.

This work presents a novel segmentation pipeline that enhances colon analysis by simultaneously segmenting air and liquid pockets within the colon. By incorporating both air and liquid regions, our approach eliminates the need for dual-position imaging, reducing patient discomfort, radiation exposure, and scan time while improving segmentation accuracy.

1.1 Key Features

- Segments both air and liquid pockets to provide a more complete representation of the colon.
- Eliminates the need for prone and supine CT scans, streamlining the imaging process.
- Improves the accuracy of virtual colonoscopy and AI-based polyp detection.
- Developed primarily for research applications, with potential utility for medical professionals in clinical practice.

1.2 Medical and Research Applications

- Virtual colonoscopy and early polyp detection.
- AI-driven colon structure analysis.
- Pre-surgical assessment and planning.
- Enhanced diagnostic capabilities in gastroenterology.

2 System Requirements

2.1 Hardware

- CPU: Any cpu with atleast 16 threads
- GPU: Any gpu with openCl compatibility
- RAM: At least 20GB recommended

2.2 Software Dependencies

- Windows 10 or 11
- To run the install script git needs to be installed
- Visual Studio 2022 with the "Desktop development with C++" and "Windows 10 SDK" packages
- ITK - Image Processing Library
- OpenCV - Computer Vision Library

2.3 Installation Instructions

To install dependencies required dependencies:

1. Execute the install.bat twice.
2. Open project in visual studio 2022 and install Desktop development with C++ and Windows 10 SDK packages if not installed.
3. In Visual Studio make sure the c++ Language standard in c++20(in Project properties - C++ - Language)
4. If missing add the following includes in Project properties - C++ - General:
   - `vcpkg\installed\x64-windows\include\ITK-5.4`
   - `vcpkg\installed\x64-windows\include\opencv4`
   - `vcpkg\installed\x64-windows\include\vxl\vcl`
   - `vcpkg\installed\x64-windows\include\vxl\core`
5. To run execute from Visual Studio

3 Implementation Details

3.1 Overview

This pipeline is designed for fully automated colon segmentation, ensuring the removal of liquid pockets that would otherwise obstruct or disrupt the colon's continuity. It follows a geometry-based approach and does not rely on deep learning models, making it robust and fully automated without the need for human intervention.

The segmentation process consists of multiple stages, as illustrated in Fig. 4:

1. **Initial segmentation:** Air-filled regions are segmented using thresholding techniques.
2. **Component filtering:** Unwanted components are removed to isolate relevant anatomical structures.
3. **Geometric boundary identification:** Flat boundary layers between air and liquid regions are detected using geometric analysis.

4. **Contrast enhancement:** Image filtering techniques are applied to enhance the contrast of high-density regions.

5. **Liquid pocket segmentation:** A region-growing algorithm is used to segment liquid pockets based on a defined intensity threshold.

6. **Integration and refinement:** The segmented air and liquid regions are merged, followed by Gaussian smoothing to produce the final segmentation output.

The code to our implementation of the pipeline is available on Github[1] there is also a slower but equivalent presentation in python.Github[2]

The next sections provide detailed explanations of each step in the pipeline.

3.2 Segmentation Algorithm

3.2.1 Thresholding for air pockets.

In this initial step, we use a straightforward thresholding operation on the unmodified Ct image with an upper bound of $-800HU$ to reliably identify all air-filled segments, including unwanted structures that must be removed.

To filter out these unwanted elements we employ a simple yet effective heuristic that doesn´t require any human interaction. Firstly, a connected components algorithm is applied with components smaller than a predefined size threshold being discarded.

We begin by removing the top and bottom slices of the image. This step is crucial because, in our observations, the lungs are truncated at the edges of these slices. As a result, both the lungs and the surrounding bed and background layers are eliminated. However, there is an exception: if a component contains very few voxels, we retain it. This is important in cases where the tube used for colon inflation appears in the bottom slice and connects to the colon component.

After this first step, the result is that the air parts of the colon are segmented; this can be only one component or many if the liquid pockets break the colon continuity.

3.2.2 Boundary Layer Detection

To effectively detect and merge the liquid pockets with the air pockets, it is crucial to identify the boundary regions between them. To achieve this, we first locate flat, downward-facing surfaces within the air pockets. These initial detections are then refined using the Sobel filter to enhance the clarity of the boundaries.

To generate the labels, we employ the Sobel operator. To speed up the process, we only compute the Sobel gradient for voxels that belong to the air segmentation. First, we calculate the Sobel filter along the Y-axis, and only if the gradient direction matches the expected boundary, do we proceed to calculate the Sobel gradients in the X and Z directions. This selective approach significantly accelerates the process compared to applying the Sobel filter across the entire image.

Next, using the Sobel values, we filter the image based on the gradient similarity to the target surface normal and group the results into connected components. A size threshold is then applied to remove small, irrelevant surfaces, leaving only the significant components, which we refer to as "flat labels."

[1]GitHub repository: https://github.com/PedroSFreire/CppSegPipeline
[2]GitHub repository: https://github.com/PedroSFreire/GeometricalColonSeg

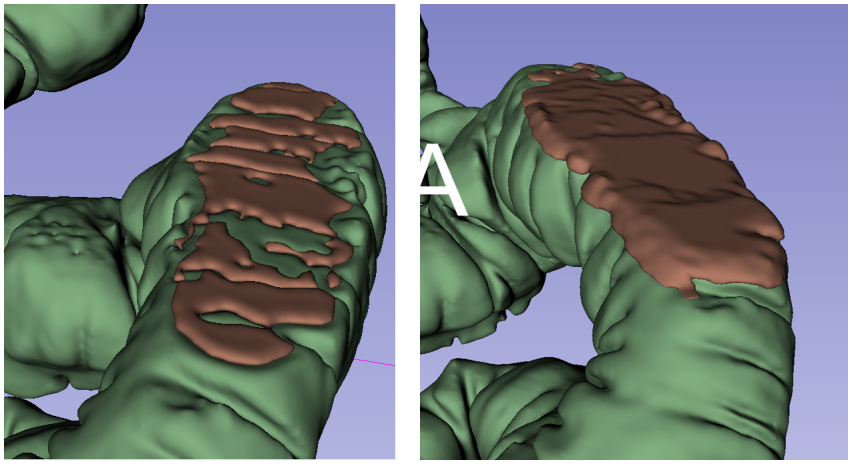

Figure 1: Reconstruction of air pockets of the colon. On the left with the flat label before expansion. On the right with the flat label region growing.

At this stage, the flat labels are still imperfect. After segmenting the liquid pockets, any labels that intersect with a liquid pocket undergo a region-growing process, which is guided by the Y-gradient direction from the Sobel filter. Labels that do not intersect with liquid pockets are discarded. To fill any gaps in the segmentation, we add any unassigned voxel located between a label voxel and a liquid pocket voxel to the label, ensuring continuity and minimising segmentation holes. The result can be seen in Fig. 1.

3.2.3 Image Filtering

Segmenting liquid pockets presents significant challenges. The contrast liquid often settles at the bottom of the pockets, making it difficult to achieve consistent segmentation(Figs. 2-right and 3-right). Additionally, low contrast between liquid and the surrounding dense organs can cause leakage into those regions. Simple thresholding may also inadvertently include bones and other dense tissues, further complicating the segmentation process.

To address this issue, we apply filters to enhance contrast before segmenting the liquid pockets. The process begins with intensity clamping, which reduces noise and isolates relevant tissue types by restricting the intensity values to a meaningful range. Next, histogram equalisation enhances contrast within the liquid pockets, making them more distinguishable. Finally, an anisotropic filter is applied to reduce noise while preserving edges, ensuring a clearer and more accurate segmentation. An example of the combined result of the filter can be seen in Fig.2.

3.2.4 Segmentation of Pockets Using Thresholding

To segment the liquid pockets we observe promising results when applying simple threshold segmentation with a lower bound of 200HU to the enhanced images to isolate the liquid pockets. However, several issues persist, many of which can be addressed with further refinement.

The primary challenge is the inclusion of unwanted structures such as bones, liquid pockets in the small bowel, and other soft tissues. To mitigate this, we leverage the previously identified flat labels from the air-filled regions of the colon.

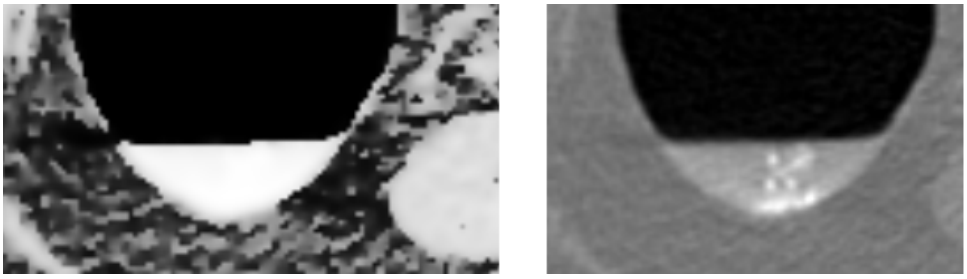

Figure 2: On the left, the contrast-enhanced ct on the right, the original. We can see the settled contrast is mostly gone.

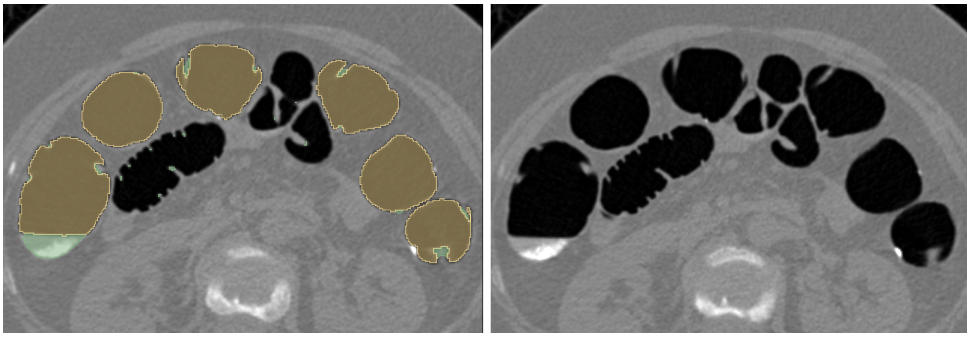

Figure 3: On the left, a segmentation of the colon is shown by a mask overlay highlighting air pockets in yellow and liquid pockets in green. This was achieved using our method. The image on the right is the unprocessed CT scan, where the heterogeneity of the contrast within the liquid pockets is more evident.

The refinement process begins by running a connected-components algorithm. Components intersecting with the previously calculated flat labels are retained, ensuring only relevant structures are included. Additionally, the detected flat labels are incorporated the segmentation to assist in defining the air-liquid boundary more accurately.

3.2.5 Intersection of Components and Boundary Region Growing

In the final step of the pipeline, we have already calculated three separate images: one representing the air pockets, another representing the liquid pockets, and the third representing the boundary regions. The goal of this step is to identify the labels that intersect both the air pockets and the liquid pockets.

Since the boundary regions are already located on the borders of the air pockets, we begin by checking for labels that intersect with a liquid pocket. If such an intersection is found, the corresponding air and liquid pockets are considered valid and included in the final segmentation image.

After confirming that the label is valid, we perform region growing on the label. This process helps refine the label and ensures the accurate representation of the combined air and liquid regions.

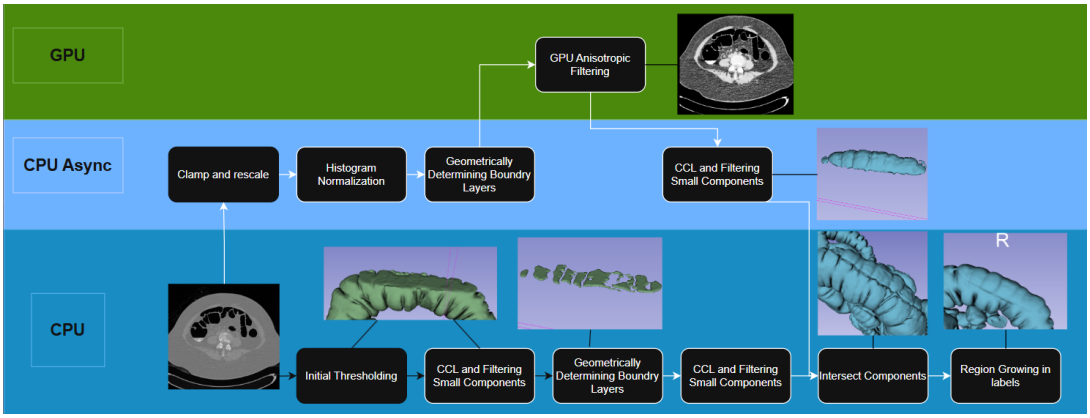

Figure 4: Optimized pipeline making use of multicore and GPU.

4 Tecnical Details

4.1 Pipeline Flow

The pipeline is divided into three main stages:

1. **Stage 1 – Air and Flat Label Detection:** This stage involves the segmentation of air-filled regions (air labels) and the identification of flat boundary regions (flat labels), followed by component-wise filtering to remove irrelevant components.

2. **Stage 2 – Liquid Pocket Segmentation:** The base image is filtered to enhance the contrast of liquid regions. These enhanced regions are then segmented into liquid pockets and filtered by connected components. This stage also includes an *anisotropic diffusion* step, which is executed on the **GPU** to improve performance due to its highly parallel nature.

3. **Stage 3 – Label Intersection and Final Region Growing:** The results from the first two stages are intersected to determine valid components. Flat labels that intersect both air and liquid regions are retained and refined via region growing to complete the final segmentation.

Stages 1 and 2 are **asynchronous** and can be executed in parallel. Synchronisation is only required before entering Stage 3 to combine the results.

Additionally, `OpenMP` (`omp.h`) is used throughout the pipeline to parallelise computational loops, effectively leveraging **multi-core CPUs** alongside the **GPU** to achieve high performance.

4.2 Code Structure and Object-Oriented Design

The code is organised using object-oriented programming (OOP) principles to enhance modularity and maintainability. The program consists of three standalone classes and two inheritance hierarchies comprising a total of eight classes the UML diagram with the classes can be in in Figure 5.

Each class is implemented across two files: a header file (`.h`) and a source file (`.cpp`), both of which follow a consistent naming convention matching the class name. For example, the class `ManualSobel` is defined in `ManualSobel.h` and implemented in `ManualSobel.cpp`.

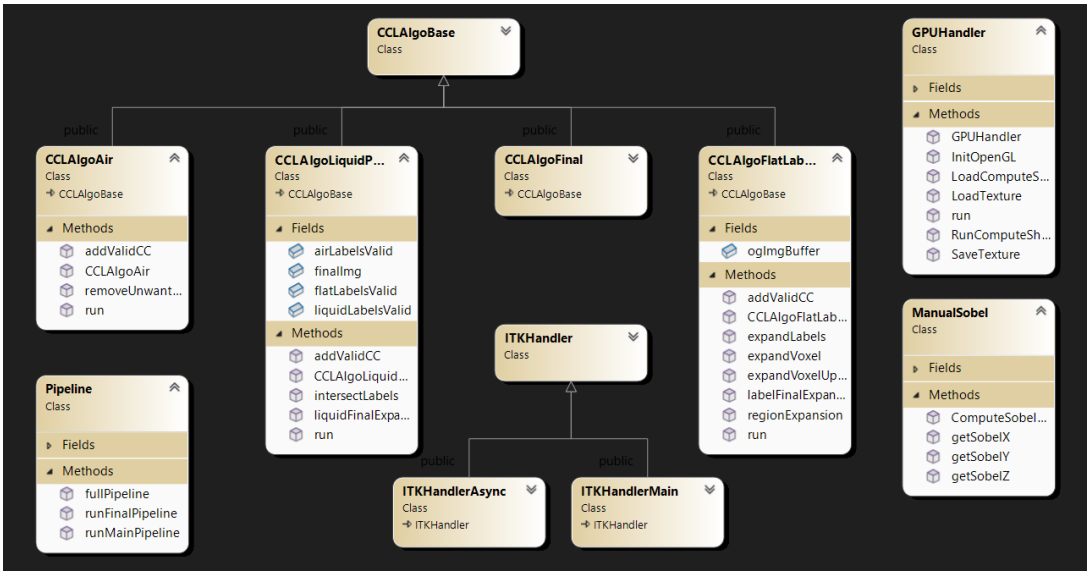

Figure 5: UML of program classes.

Pipeline Class The central class of the program is `Pipeline`, which manages the core logic of the segmentation process. It provides three main functions:

- `runMainPipeline()`: Executes the first stage of the pipeline, responsible for generating the air and flat labels.
- `runFinalPipeline()`: Handles the final stage, including component intersection and region growing of the valid flat labels.
- `fullPipeline()`: Orchestrates the entire process by calling the functions above in the correct sequence. It also manages the asynchronous execution of the liquid segmentation stage and ensures proper synchronisation before the final stage.

ManualSobel Class The `ManualSobel` class serves primarily as a utility library for efficient Sobel gradient computation. Unlike ITK's built-in Sobel operator, which computes gradients across the entire image and all three axes, `ManualSobel` allows for the targeted calculation of the Sobel gradient at a specific voxel and along a specified axis. This significantly improves performance, particularly in use cases where gradient information is only needed for a subset of voxels.

GPUHandler Class The `GPUHandler` class is responsible for performing anisotropic filtering on the GPU. Instead of loading shader code from an external file, the GLSL compute shader is embedded directly as a string within the class. `GPUHandler` encapsulates all GPU-related operations, including shader compilation, buffer management, execution, and synchronisation. This design ensures that GPU processing is modular and well-contained within a single class.

CCLAlgo Inheritance Hierarchy The `CCLAlgo` inheritance hierarchy is composed of five classes, centred around a shared base class, `CCLAlgoBase`. This base class defines the common data structures and functionality required for connected component labelling across multiple segmentation stages.

CCLAlgoBase The CCLAlgoBase introduces and manages a core data structure: a `std::vector` of `ccData` structures. Each `ccData` contains:

- The size of the connected component (i.e., the number of voxels it includes),
- A `std::vector<int>` listing all voxel indices that belong to the component.

This base class provides several key methods:

- `runCCL()`: Reads the image and fills the component data structures through connected component labelling.
- `removeCC(index)`: Removes a specific component.
- `removeSmallCC(minSize) and removeSmallBigCC(maxSize)`: Eliminates all components smaller or larger than a specified size threshold.

`CCLAlgoBase` is never instantiated directly; it serves purely as a foundation for the following four specialised subclasses:

CCLAlgoAir The `CCLAlgoAir` class inherits from `CCLAlgoBase` and provides the necessary logic for segmenting air-filled regions in the CT scan. It includes functionality to:

- Remove unnecessary components such as lungs, background, and bed regions — primarily through analysis of top and bottom image slices.
- Add the air pocket components that were marked as valid during the final intersection step into the final segmentation label.

CCLAlgoFlatLabels The `CCLAlgoFlatLabels` class also inherits from `CCLAlgoBase` and supports flat surface region labelling. It includes methods for:

- Vertical expansion of flat labels to better connect air-liquid interface regions.
- Final region growing of valid flat labels during the last stage of the pipeline.

Note that the creation of the flat labels themselves is handled outside this class, specifically in `ITKHandlerMain`, as it requires image and gradient data only available there. This class is responsible for adding the valid flat labels (identified through final intersection) to the final segmentation output.

CCLAlgoLiquid `CCLAlgoLiquid` extends `CCLAlgoBase` and handles the final-stage region growing for liquid pockets. This step is intended to patch small holes that might remain after initial expansion while avoiding leakage into surrounding tissues. This class also:

- Adds selected liquid pocket components (validated in the final intersection step) to the output label.
- Performs the actual union of air, liquid, and flat label components into the composite intersection image.

CCLAlgoFinal The `CCLAlgoFinal` class is the final step in the hierarchy. It processes the combined intersection result from all component types and identifies the primary colon segment. This is done by selecting the component that contains the voxel with the lowest Y-coordinate in the image (typically corresponding to the lowest physical point), which is assumed to belong to the colon. This component is then selected for inclusion in the final output label.

ITKHandler Inheritance Hierarchy The `ITKHandler` inheritance hierarchy is composed of three classes, with `ITKHandler` serving as the shared base class. This base class provides a unified interface and shared functionality for both the synchronous and asynchronous stages of the pipeline. It encapsulates the common ITK filters and utility functions used across different phases of the segmentation process.

ITKHandlerMain `ITKHandlerMain` extends `ITKHandler` and adds specific functionality related to the first stage of the pipeline. It is responsible for:

- Setting up and executing the initial thresholding operations used to segment air-filled regions.
- Computing the flat boundary labels used in later stages to detect air-liquid interfaces.

This class is executed synchronously and is designed to operate in parallel with the asynchronous components of the pipeline.

ITKHandlerAsync `ITKHandlerAsync` also inherits from `ITKHandler` and handles the tasks associated with the second, asynchronous stage of the pipeline. It includes:

- Execution of a custom histogram-based image filter to improve visibility and contrast of liquid-filled pockets.
- Integration with the `GPUHandler` class to apply GPU-based anisotropic diffusion filtering for enhanced segmentation performance.
- Initialisation and management of the asynchronous thread that runs in parallel with the main processing thread.

The results from this class are synchronised in the `Pipeline` class before entering the final stage of the pipeline.

5 Usage Guide

Once the program is installed, it can be run from Visual Studio in both Debug and Release configurations. The Release mode provides significantly better performance.

Input Image Requirements This program is designed to process abdominal CT scans of an inflated colon with dietary preparation and contrast ingestion. It was developed specifically to segment images from the Cancer Imaging Research Archive (TCIA)[3]. However, some images in this dataset do not include contrast ingestion, and due to the

[3]Dataset repository: https://www.cancerimagingarchive.net/access-data/

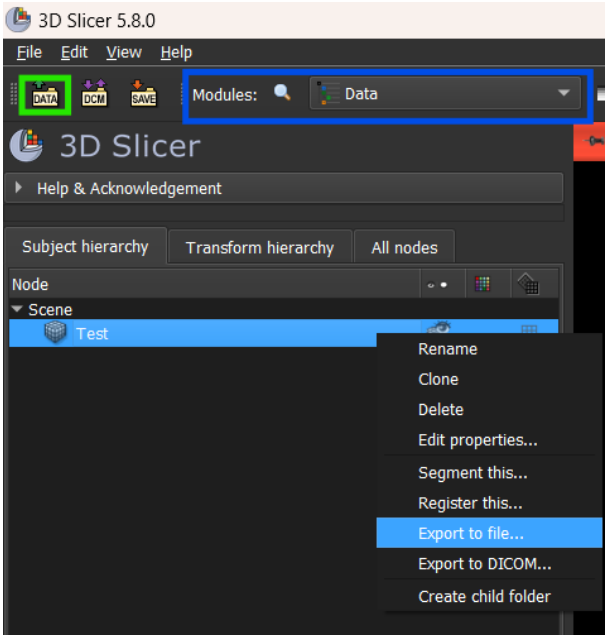

Figure 6: 3d slicer GUI to export images as .nii.

low contrast of liquid pockets in such cases, the segmentation pipeline may not work effectively.

Image Format and Conversion The Cancer Imaging Research Archive (TCIA) provides images in .dcm (DICOM) format, while this program requires .nii (NIfTI) format. To convert DICOM images to NIfTI, you can use 3D Slicer[4] by following these steps:

- Open 3D Slicer.
- Go to Data(green box6) → Choose Directory to Add and select the folder containing the .dcm files.
- Open Subject Hierarchy panel with the drop down menu(blue box6) and selecting data, right-click the loaded image and select Export to File.
- Choose .nii as the output format and save the file.

6 Conclusion

This documentation has presented a comprehensive overview of a fully automated colon segmentation pipeline developed for processing abdominal CT scans. The pipeline was specifically designed to handle datasets with dietary contrast ingestion and colon inflation, enabling the simultaneous segmentation of both air and liquid pockets—thereby removing the need for dual-position scanning protocols.

The implementation emphasises performance and scalability by combining multi-threaded CPU processing using OpenMP with GPU-accelerated anisotropic filtering through compute shaders. The pipeline is structured into clearly defined stages and

[4]3D Slicer: https://www.slicer.org/

organised using object-oriented programming principles, providing modularity and ease of extension. Key features include efficient custom Sobel filtering, geometrically guided flat label detection, and region-growing algorithms that ensure segmentation continuity even in some challenging anatomical cases.

7 Future work

The asynchronous thread completes significantly faster than the main thread. This idle time presents an opportunity to enhance segmentation quality without impacting overall performance. For instance, an early liquid pocket expansion step could be added here to better capture challenging liquid pockets, ensuring higher-quality inclusion in the final segmentation. Additionally, this spare time could be used to apply a more comprehensive filter stack, further improving the definition of liquid pockets all at no extra performance cost.