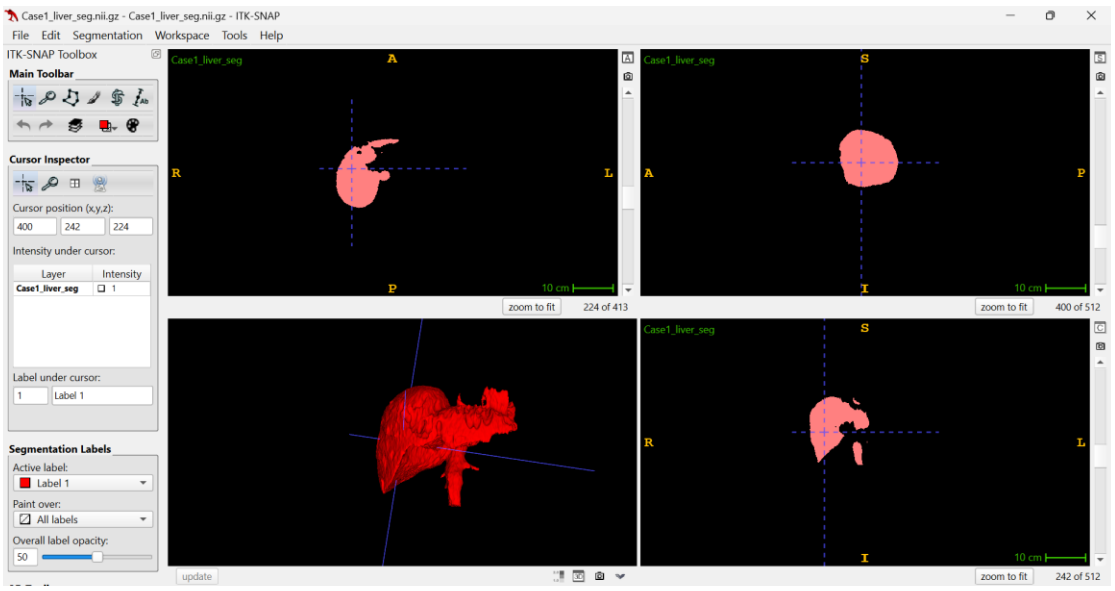
Medical Image Processing (67705):

Exercise 1- Part 2 – Liver Segmentation

By: Amit Halbreich, ID: 208917393

CT Case 1 — Liver Segmentation Result



Case1_liver_seg.nii.gz - Case1_liver_seg.nii.gz - ITK-SNAP

Cursor position (x,y,z): 400 242 224

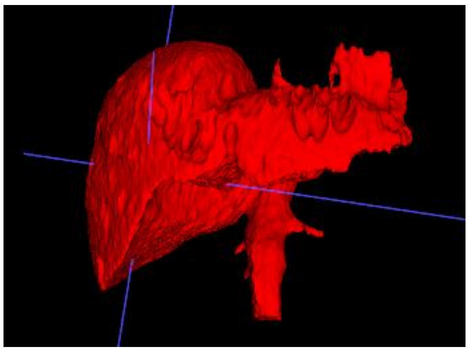
Layer	Intensity
Case1_liver_seg	1

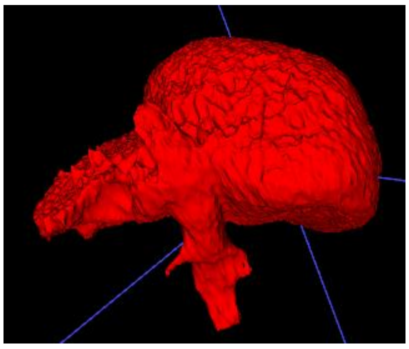
Label under cursor: 1 Label 1

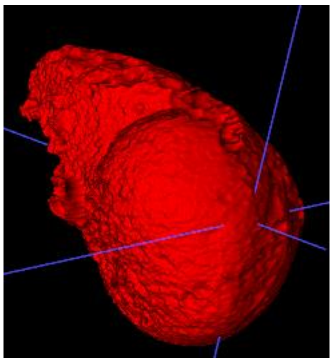
Active label: Label 1

Paint over: All labels

Overall label opacity: 50







Ground Truth File:

MIP Data/Case1_liver_segmentation.nii.gz

Calculated Liver Segmentation:

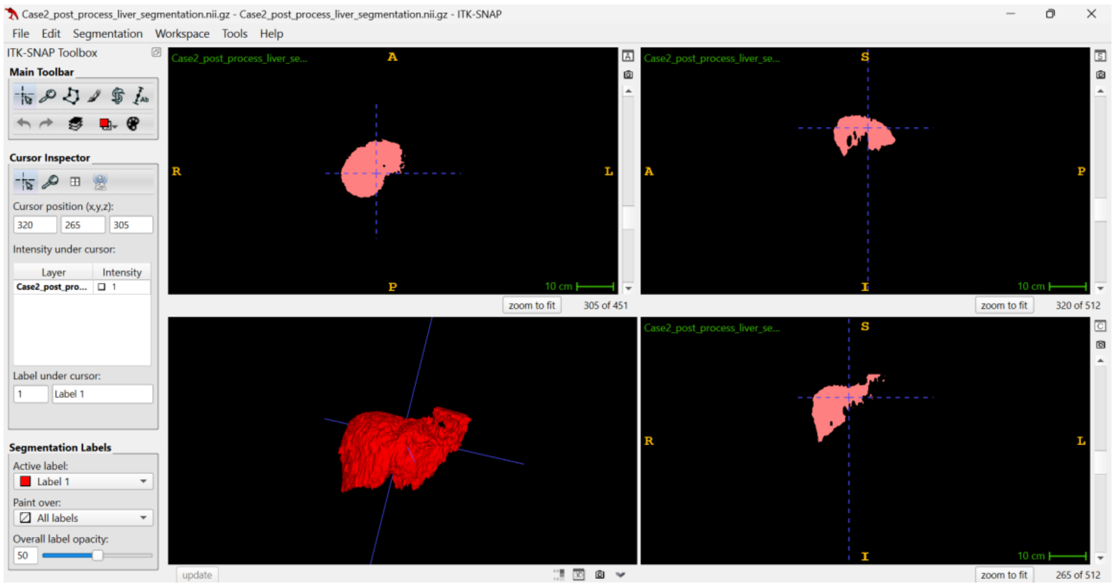
MIP Data/Segmentations/Case1_liver_seg.nii.gz

DICE Score & VOD Score

$$DICE = 0.9101813896660667$$

$$VOD = 0.16483221975152706$$

CT Case 2 — Liver Segmentation Result



Case2_post_process_liver_segmentation.nii.gz - Case2_post_process_liver_segmentation.nii.gz - ITK-SNAP

Cursor position (x,y,z):

320	265	305

Intensity under cursor:

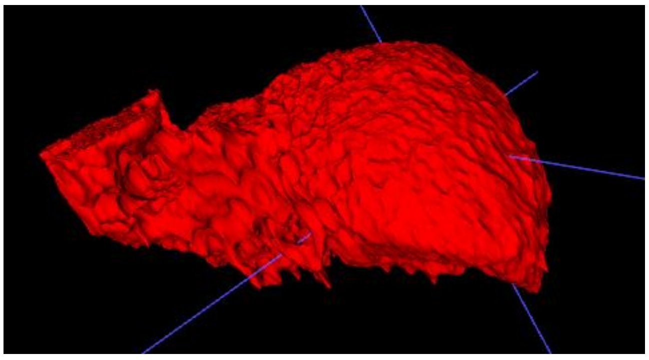
Layer	Intensity
Case2_post_pro...	1

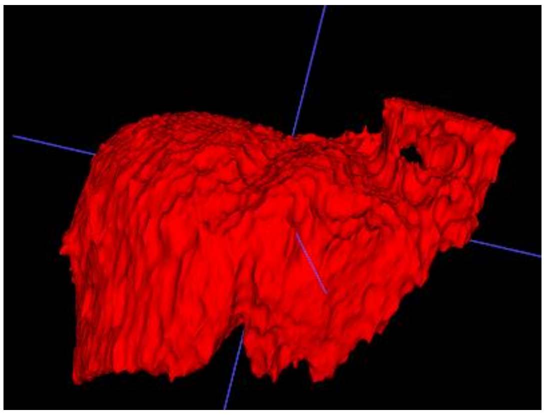
Label under cursor: 1 Label 1

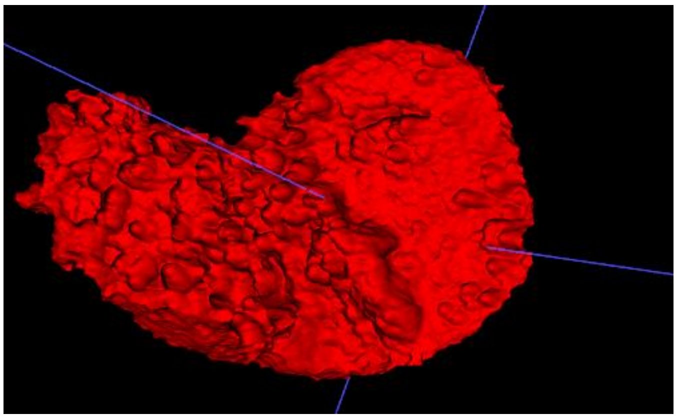
Active label: Label 1

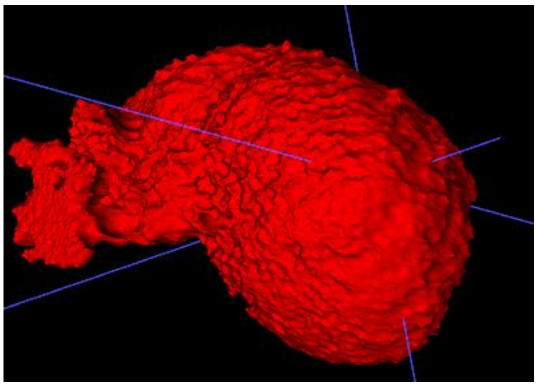
Paint over: All labels

Overall label opacity: 50

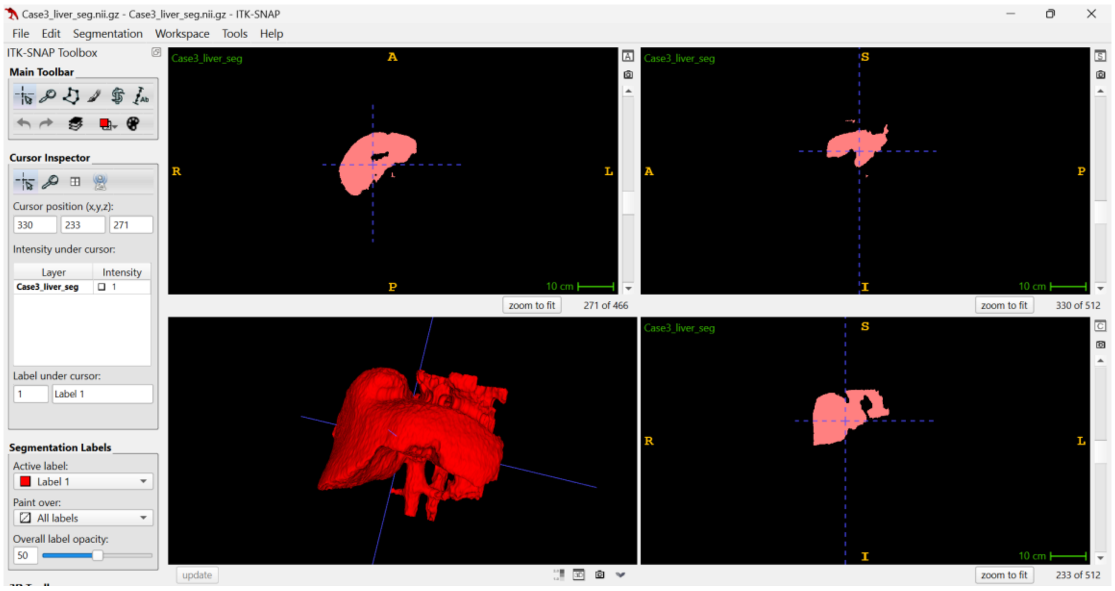








CT Case 3 – Liver Segmentation Result



Cursor position (x,y,z): 330 233 271

Intensity under cursor:

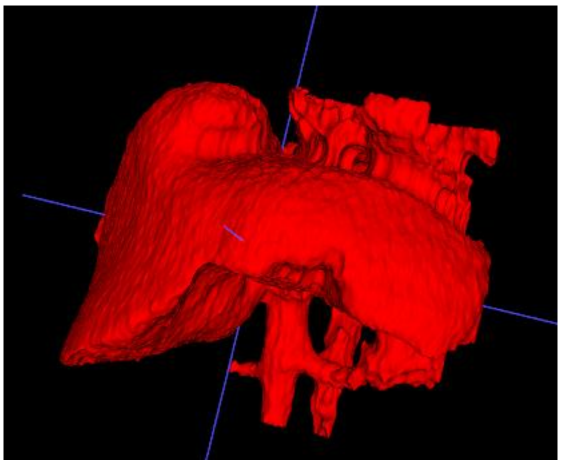
Layer	Intensity
Case3_liver_seg	1

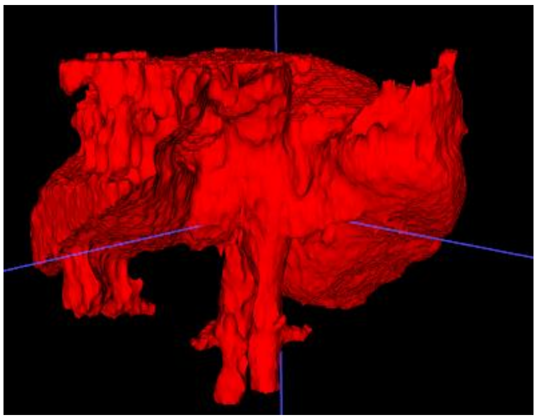
Label under cursor: 1 Label 1

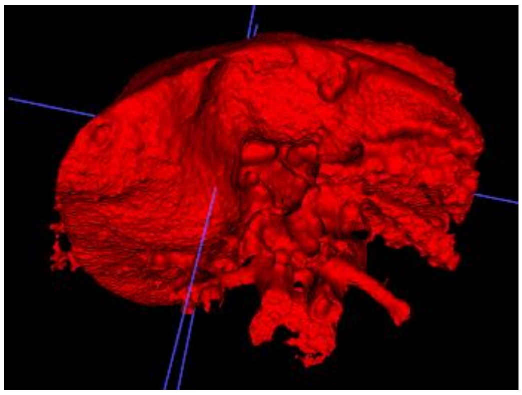
Active label: Label 1

Paint over: All labels

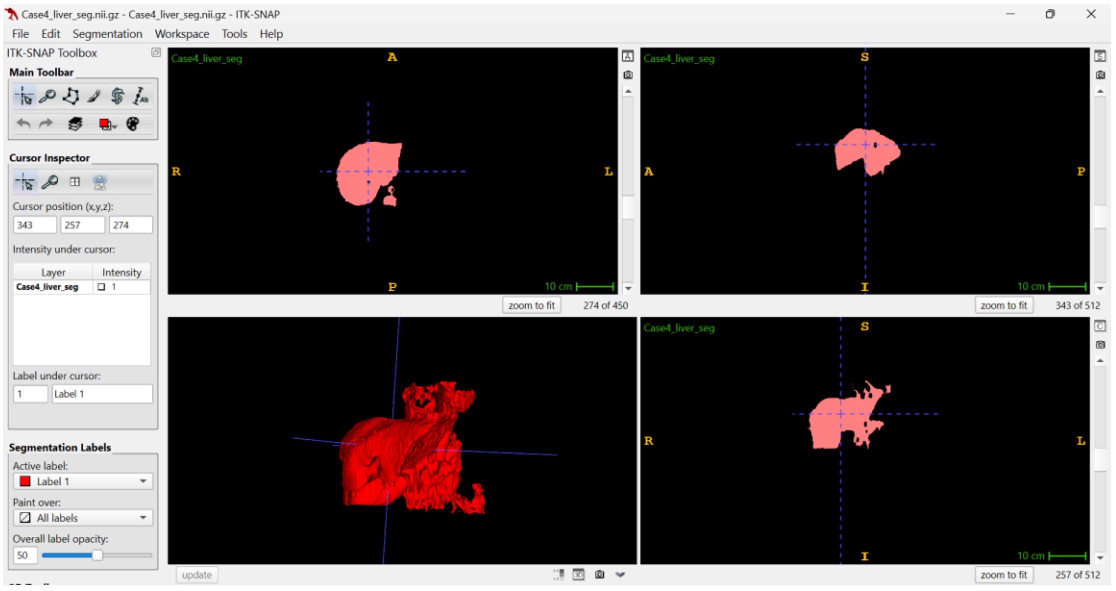
Overall label opacity: 50







CT Case 4 — Liver Segmentation Result


Case4_liver_seg.nii.gz - Case4_liver_seg.nii.gz - ITK-SNAP

Cursor position (x,y,z): 343 257 274

Intensity under cursor:

Layer	Intensity
Case4_liver_seg	1

Label under cursor: 1 Label 1

Active label: Label 1

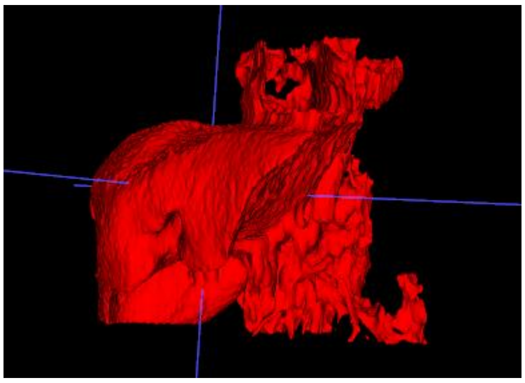
Paint over: All labels

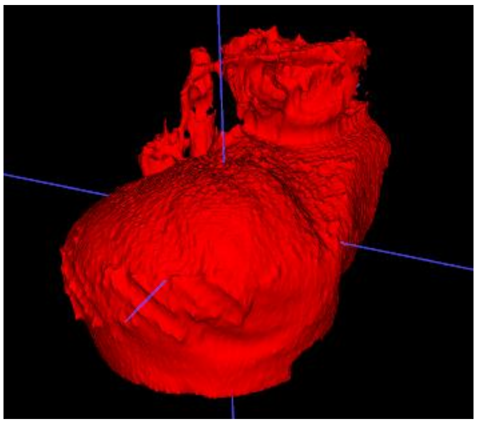
Overall label opacity: 50

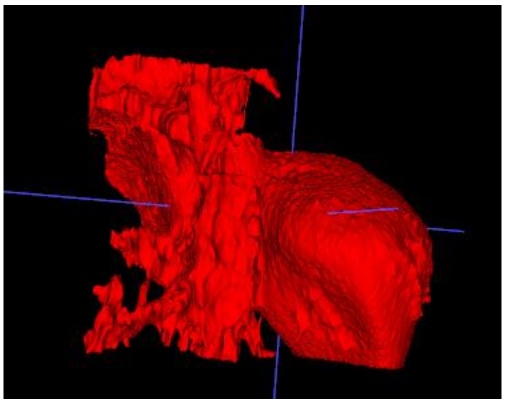
274 of 450

343 of 512

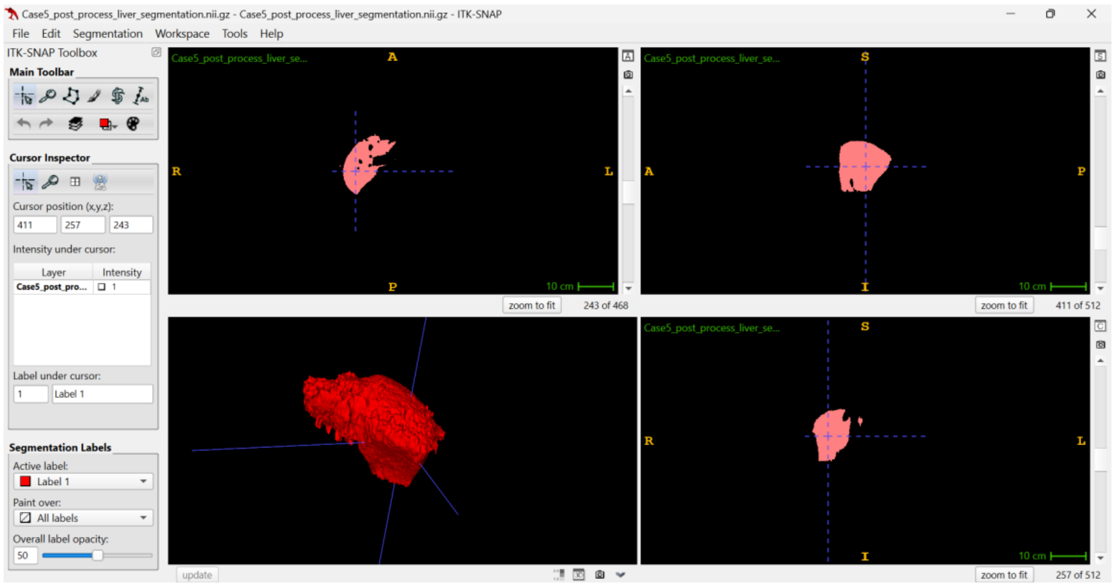
257 of 512







CT Case 5 — Liver Segmentation Result


Case5_post_process_liver_segmentation.nii.gz - Case5_post_process_liver_segmentation.nii.gz - ITK-SNAP

Cursor position (x,y,z): 411 257 243

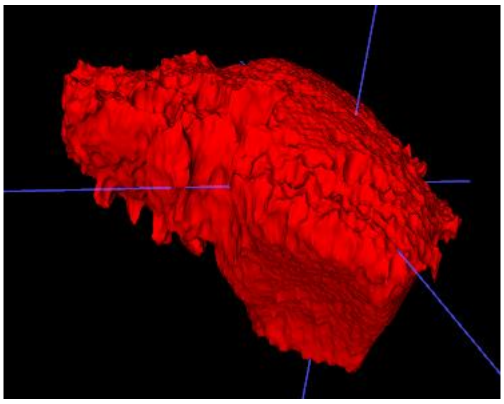
Layer	Intensity
Case5_post_pro...	1

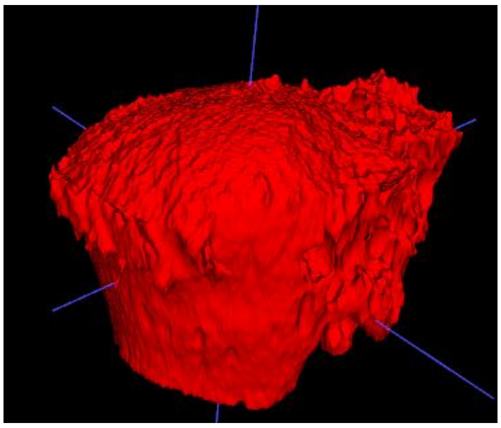
Label under cursor: 1 Label 1

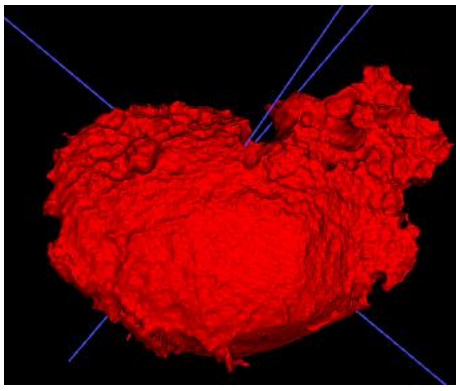
Active label: Label 1

Paint over: All labels

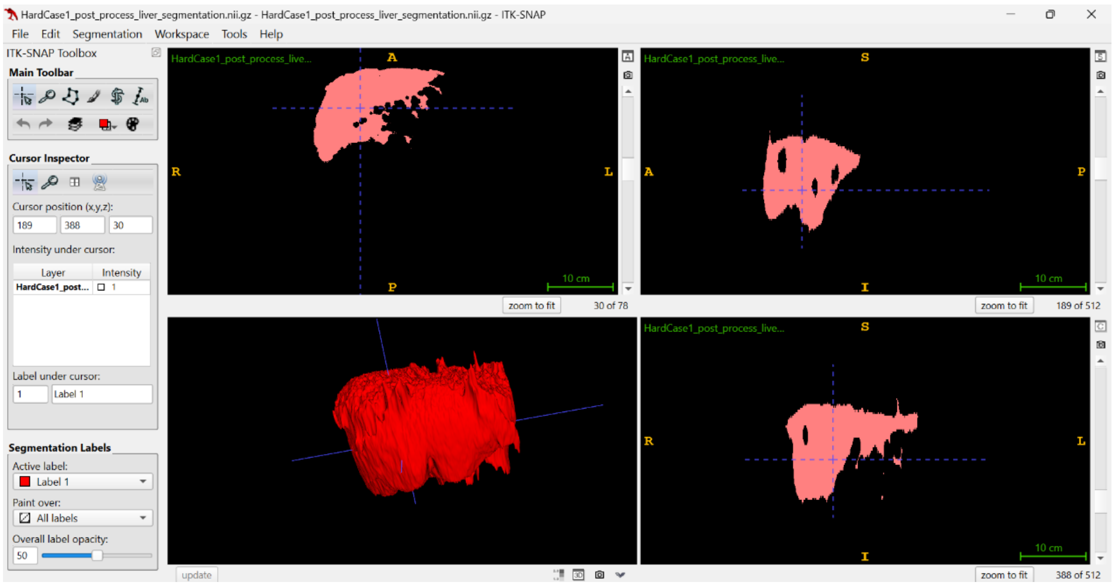
Overall label opacity: 50

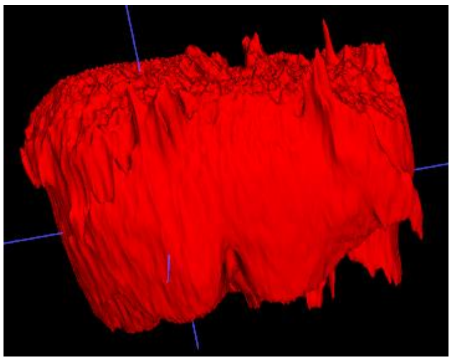


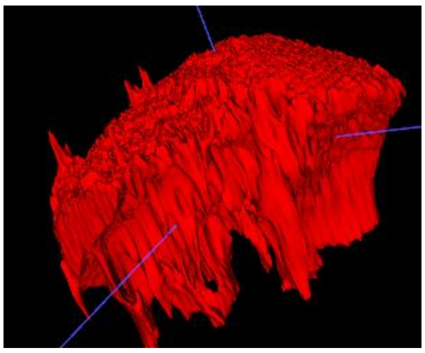


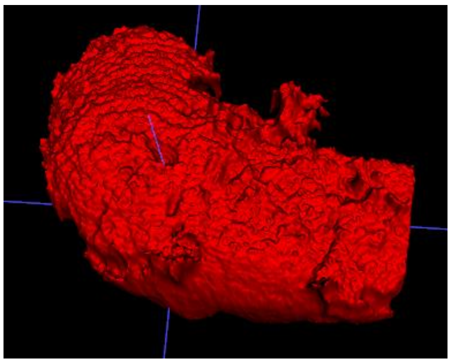


CT Hard Case 1 — Liver Segmentation Result









<u>Ground Truth File:</u>

MIP Data/HardCase1_liver_segmentation.nii.gz

<u>Calculated Liver Segmentation:</u>

MIP Data/Segmentations/HardCase1_post_process_liver_segmentation.nii.gz

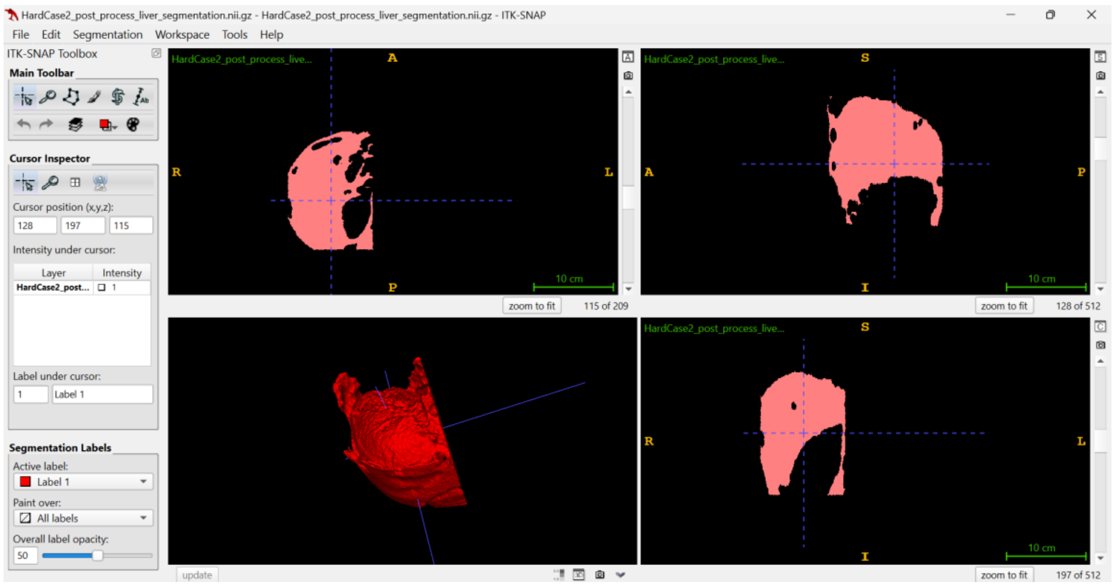
<u>DICE Score & VOD Score</u>

$$DICE = 0.8804743206954235$$

$$VOD = 0.21352914276843227$$

CT Hard Case 2 – Liver Segmentation Result

As we can spot the cleaning process was a little harder for this case – even after taking "tight" boundaries for the liver ROI. Some of the skin and other organs were added maybe because they are somewhat close in their HU Unit value.


HardCase2_post_process_liver_segmentation.nii.gz - HardCase2_post_process_liver_segmentation.nii.gz - ITK-SNAP

Cursor position (x,y,z): 128 197 115

Intensity under cursor:

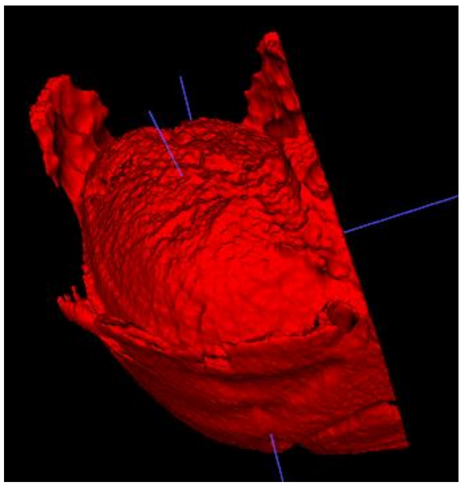
Layer	Intensity
HardCase2_post...	1

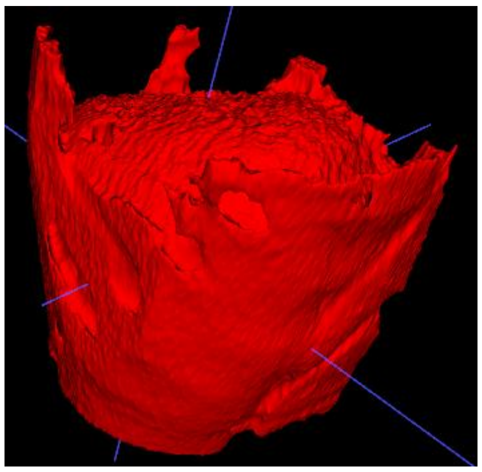
Label under cursor: 1 Label 1

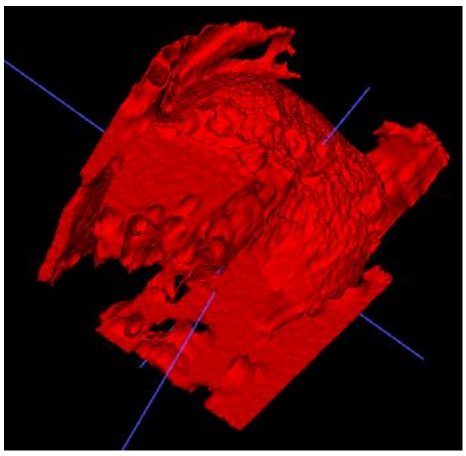
Active label: Label 1

Paint over: All labels

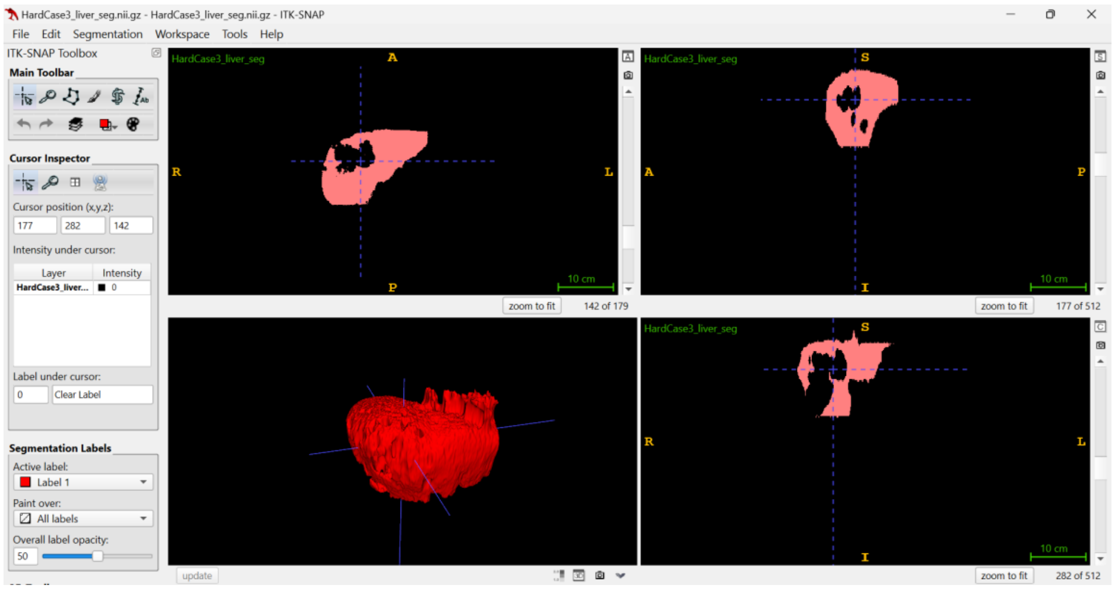
Overall label opacity: 50







CT Hard Case 3 — Liver Segmentation Result


HardCase3_liver_seg.nii.gz - HardCase3_liver_seg.nii.gz - ITK-SNAP

Cursor position (x,y,z): 177 282 142

Intensity under cursor:

Layer	Intensity
HardCase3_liver...	■ 0

Label under cursor: 0 Clear Label

Segmentation Labels

Active label: Label 1

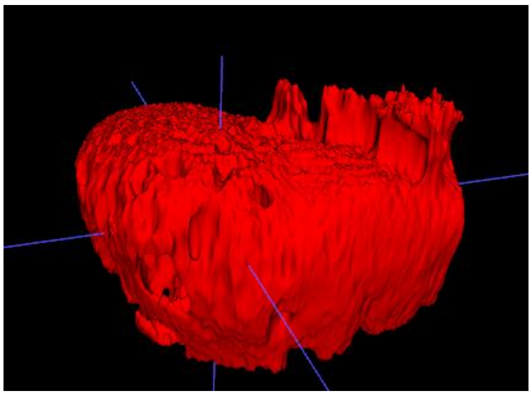
Paint over: All labels

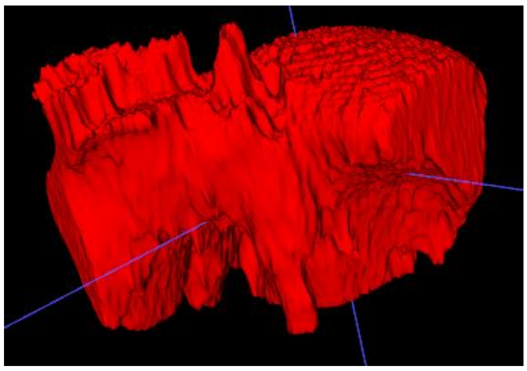
Overall label opacity: 50

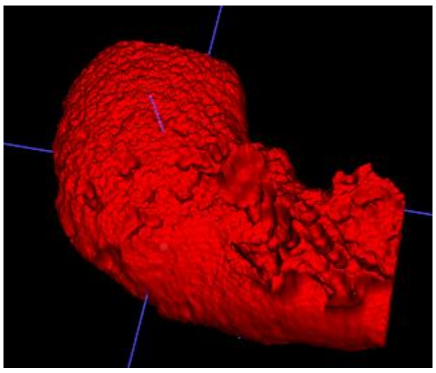
142 of 179

177 of 512

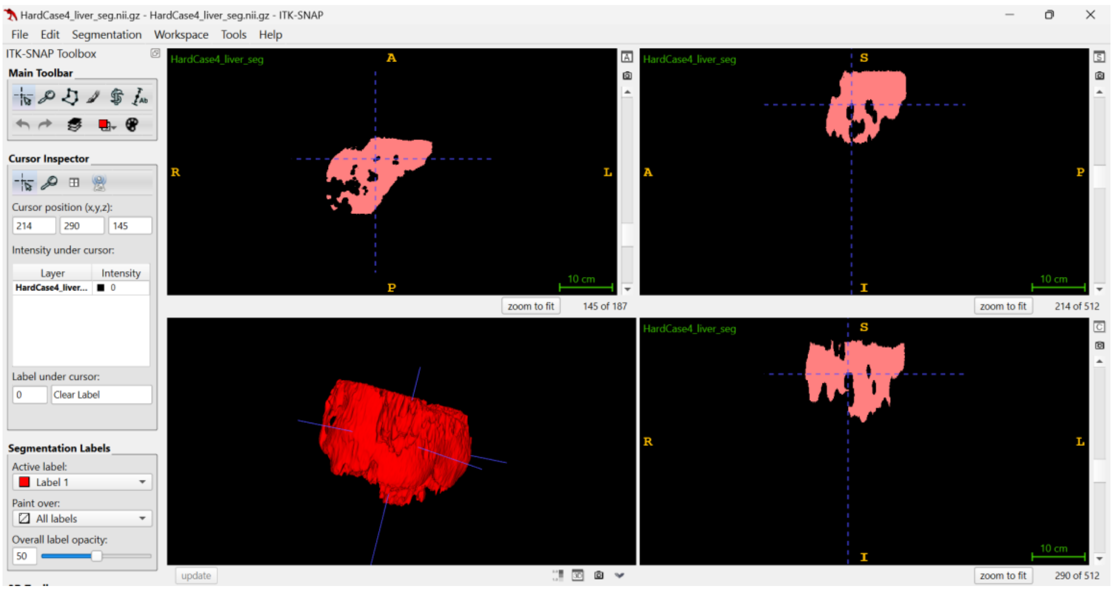
282 of 512

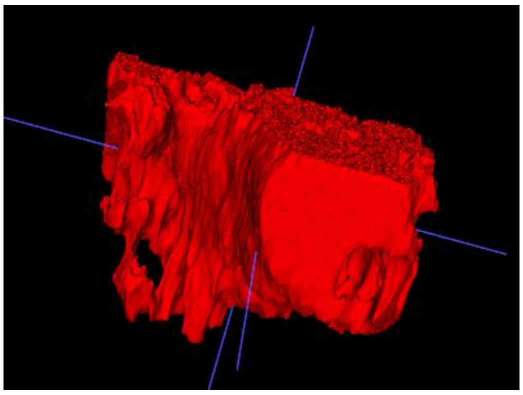


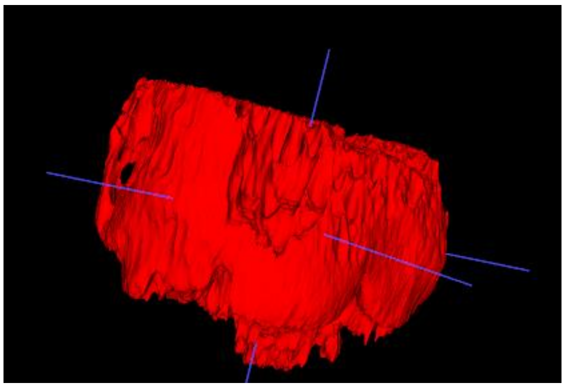


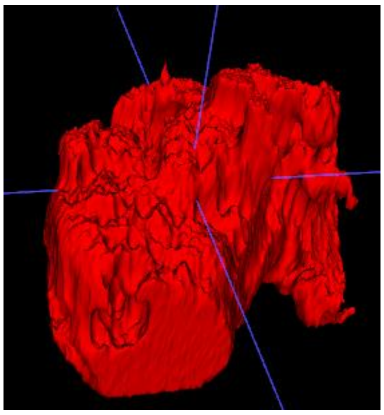


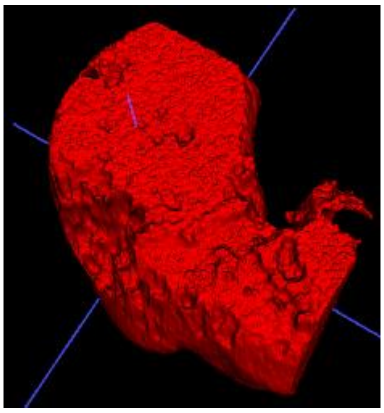
CT Hard Case 4 — Liver Segmentation Result





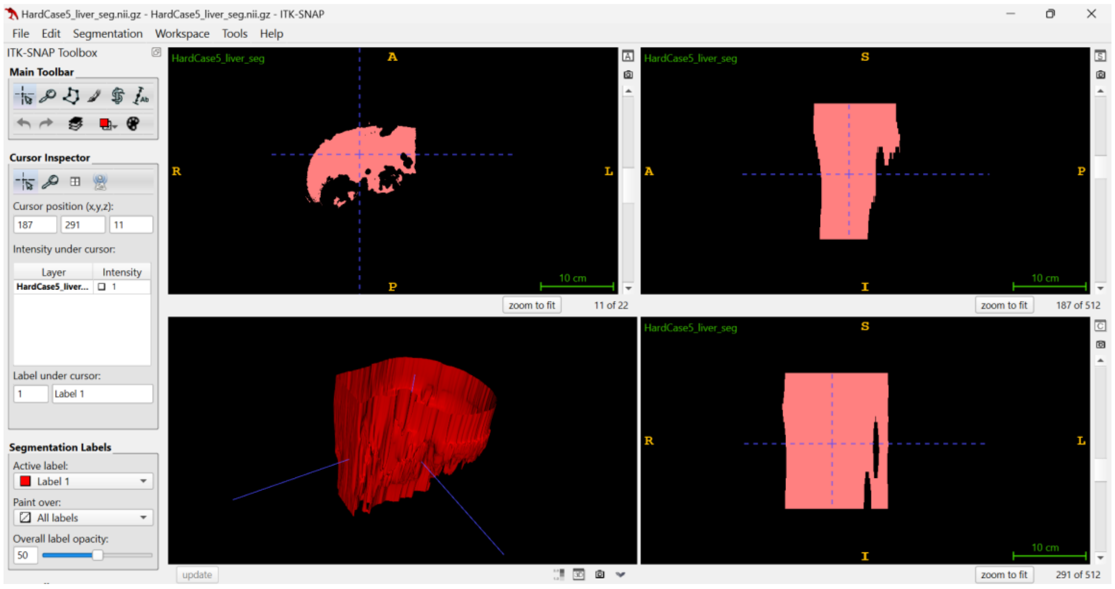






CT Hard Case 5 — Liver Segmentation Result

As we can spot because there are only 22 axial slices in this CT scan it has very low resolution but the shape of the Liver Segmentation resembles the triangular anatomical shape of the liver. And the axial slice of the segmentation on the top-right image resembles an actual liver axial slice.



Cursor position (x,y,z): 187 291 11

Intensity under cursor:

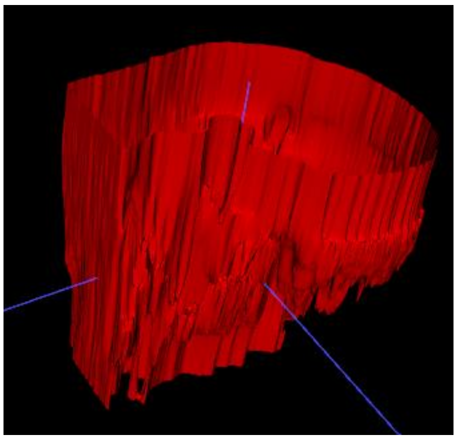
Layer	Intensity
HardCase5_liver...	1

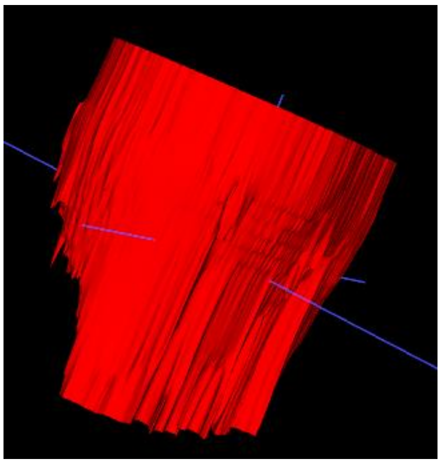
Label under cursor: 1 Label 1

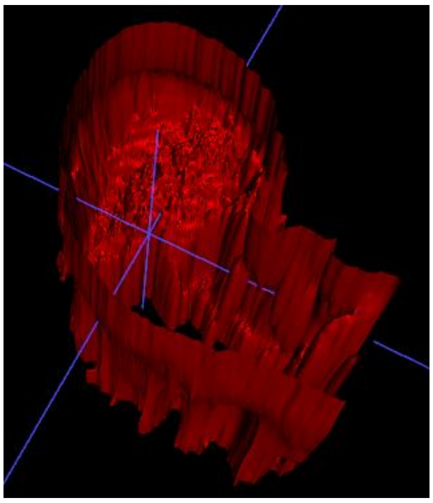
Active label: Label 1

Paint over: All labels

Overall label opacity: 50







Functions & Explanations

Code Style & Insights

Use of Vectorial and Matrixial Coding:

NumPy Arrays: The code utilizes NumPy arrays for efficient handling of multidimensional data, enabling vectorized operations.

Vectorized Operations: Vectorized operations are extensively used throughout the code for tasks like thresholding, finding voxel neighbors, computing gradient magnitudes, running Multi Seeded Region Growing algorithm with matrixial and vectorized data – taking advantage of taking all seeds and their neighbors to the queue and updating it in each iteration by taking the new front voxels that hasn't been visited yet – all this for enhancing performance by avoiding explicit loops.

Matrix Operations: Matrices are used to represent voxel coordinates and masks, facilitating efficient operations like element-wise multiplication and logical indexing.

Optimized Algorithms: The MSRG algorithm is implemented efficiently using matrix operations and vectorized logic, reducing redundant computations and improving runtime performance.

Results and Performance:

Segmentation Accuracy: The segmentation accuracy is evaluated using DICE and VOD scores, comparing the segmented liver with ground truth data. According to the Formula:

$$DICE = \frac{2 \cdot |VOL(GT\_Segmentation) \cap VOL(Estimated\_Segmentation)|}{|VOL(GT\_Segmentation)| + |VOL(Estimated\_Segmentation)|}$$

$$VOD = 1 - \frac{|VOL(GT\_Segmentation) \cap VOL(Estimated\_Segmentation)|}{\left|VOL\left(GT_{Segmentation}\right) \cup VOL(Estimated\_Segmentation)\right|}$$

Runtimes: The emphasis on vectorial and matrixial coding helps in achieving shorter runtimes, especially for computationally intensive tasks like MSRG, ensuring reasonable performance even for large nifti files.

Scalability: The code design allows for scalability, enabling it to handle different CT scan sizes efficiently.

Output: The code saves segmented liver data and evaluation results, providing insights into segmentation quality and aiding further analysis of different stages in the liver segmentation process.

Imported Library Functions

```python
import os
import random

import numpy as np
import nibabel as nib
from skimage.morphology import binary_opening, binary_closing, ball,
label
from skimage.measure import label
```

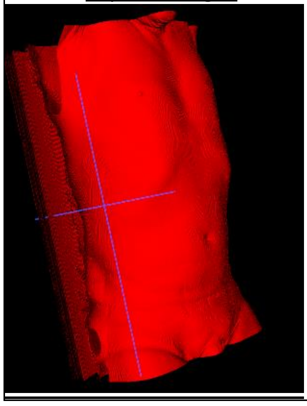
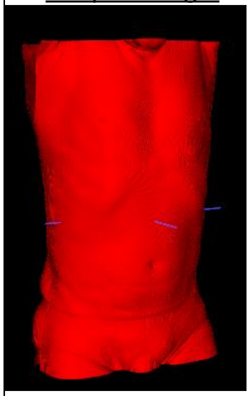
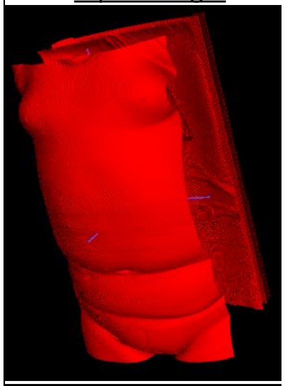
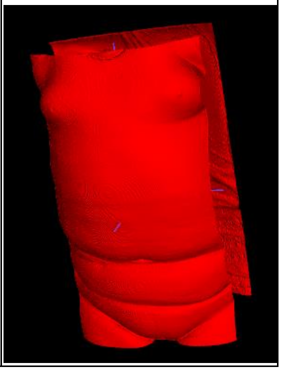
- I used skimage.morphology .binary_opening and skimage.morphology.binary_closing for operations for post_processing or pre-processing data before performing binary threshold or get the largest Connected Component. skimage.morphology.ball is used as binary opening and binary closing object to dilate or erose.
- I used numpy library for operations and calculations.
- I used nibabel library in order to load,save and read nifty files.
- I used skimage.measure.label library function in order to get labeled scans and get the number of Connected Components to make sure I get 1 CC after body isolation process etc.

Nifti files Utils – Helper Functions

```
# ---------------------------------------------------------------------
----------
# Part 1:  Helper Functions - Nifti Utils

def load_nifti(image_path):
    """
    This function loads a Nifti image file.
    :param image_path: The path to the Nifti image file.
    :return: The image data, the image filename, and the Nifti file
object.
    """
    nifti_file = nib.load(image_path)
    img_data = nifti_file.get_fdata()
    return img_data, image_path, nifti_file


def save_nifti(img_data, ct_filename_path, output_path):
    """
    This function saves Nifti image data to a file.
    :param img_data: The image data to be saved.
    :param ct_filename_path: The path to the original CT filename.
    :param output_path: The path where the Nifti image will be saved.
    """
    new_nifti = nib.Nifti1Image(img_data,
nib.load(ct_filename_path).affine)
    nib.save(new_nifti, output_path)


def create_folder(output_folder):
    """
    This function creates a folder at the specified output path if it
doesn't
    already exist.
    :param output_folder: The path of the folder to be created.
    """
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)


def save_scan_data_to_path(img_data, output_path, file_name):
    """
    This function saves the scan data to the specified output path.
    :param img_data: The scan data to be saved.
    :param output_path: The path where the scan data will be saved.
    :param file_name: The file name of the original CT scan.
    """
    new_nifti = nib.Nifti1Image(img_data.astype(np.uint8),
                                nib.load(file_name).affine)
    nib.save(new_nifti, output_path)

# ---------------------------------------------------------------------
----------
```

Isolate Body

```
Function Name: def isolate_body(ct_data,
body_lower_threshold=BODY_LOWER_THRESHOLD,
                 body_upper_thresh=BODY_UPPER_THRESHOLD,
                 ball_size=2, ct_path=''):
    """
    This function isolates the body from CT data and separates
between the
    patient's body mask and the bed of the CT chamber. At the end the
noise
    from the image is cleaned.
    :param ct_data: The CT data.
    :param body_lower_threshold: The lower threshold for the body.
    :param body_upper_thresh: The upper threshold for the body.
    :param ball_size: The size of the ball structuring element.
    :param min_size_obj_range: The range of objects to remove.
    :return: The segmented isolated body.
    """
```

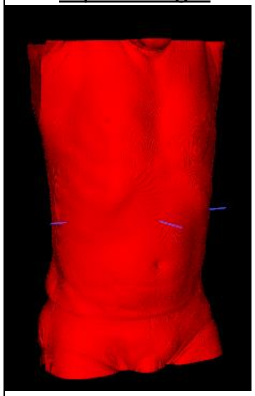
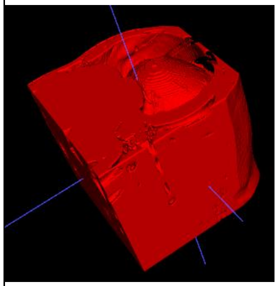
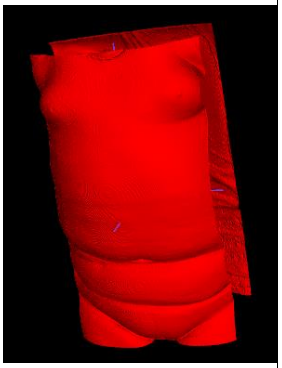
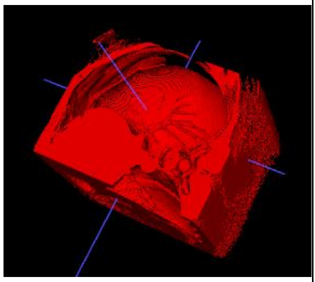
Input: ct_scan (Hard Case 1-5 or Case 1-5)			
Output: body_mask with patient's body separated from the bed (for Hard Case 1-5 or Case 1-5) – We can spot the input images has the patients' beds in it and after body isolation we remain only with body segmentation without the bed or noise.			
Input Image:	Output Image:	Input Image:	Output Image:
			

Create Liver ROI

```
Function Name: def create_liver_roi(ct_data, aorta_data,
                      liver_lower_th=LIVER_MIN_THRESHOLD,
liver_upper_th=LIVER_MAX_THRESHOLD,
                      liver_roi_x=(-30, 195), liver_roi_y=(-100, 100),
                      liver_roi_z=(-75, 50), ct_path=''):
    """
    This function creates a region of interest (ROI) for the liver.
    :param ct_data: The CT data.
    :param aorta_data: The aorta data.
    :param liver_lower_th: The lower threshold for the liver.
    :param liver_upper_th: The upper threshold for the liver.
    :param liver_roi_x: The X-axis range for the liver ROI.
    :param liver_roi_y: The Y-axis range for the liver ROI.
    :param liver_roi_z: The Z-axis range for the liver ROI.
    :param ct_path: The path to the CT file.
    :return: The liver region of interest.
    """
```

Input: ct_data, aorta data (Hard Case 1-5 or Case 1-5)

Output: liver_roi a bounding box that approximately finds the liver's region in the ct_scan by calculating the bounding box of the ROI from the Aorta given segmentation and extracting its bounding box (for Hard Case 1-5 or Case 1-5)

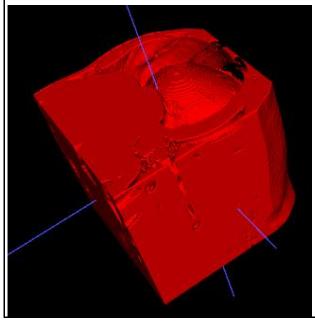
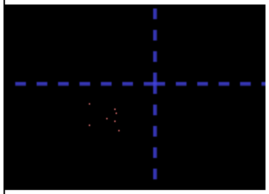
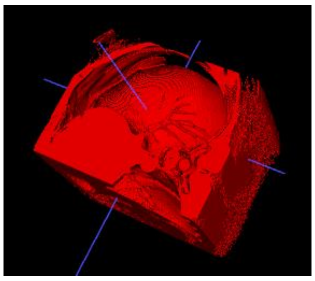
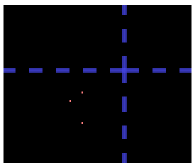
Input Image:	Output Image:	Input Image:	Output Image:
			

Find Seeds

```
Function Name: find_seeds(roi_data, num_seeds=NUM_SEEDS):
    """
    This function finds seeds for region growing.
    :param roi_data: The region of interest data.
    :param num_seeds: The number of seeds to find.
    :return: The seeds.
    """
```

Input: roi_data, num_seeds (Hard Case 1-5 or Case 1-5)

Output: seeds – finds seeds with lowest gradients inside the liver ROI, then selects from the ROI seeds with lowest gradients magnitude $(\partial\|x^2\|, \partial\|y^2\|, \|z^2\|)$ randomly from the list of low gradients magnitudes – Below we can see the seeds as small dots selected from ROI

Input Image: **CT Case 1 – Liver ROI**	Output Image: **CT Case 1 – Seeds**	Input Image: **CT Case 2 – Liver ROI**	Output Image: **CT Case 2 – Seeds**
			

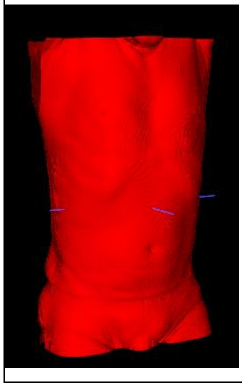
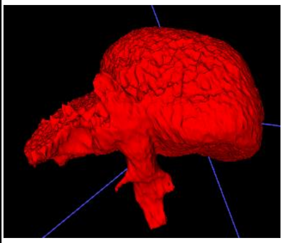
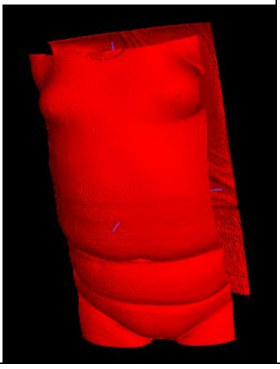
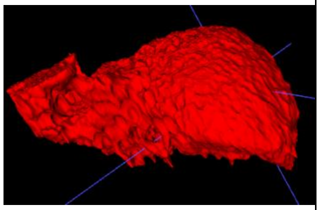
Multiple Seeds Region Growing

```
Function Name: def multiple_seeds_RG(ct_data, roi_data,
max_iter=MAX_ITERS, ct_path=''):
    """
    Perform Multi-Seeded Region Growing algorithm fully with all
parts.

    Args:
        ct_data (ndarray): The CT data.
        roi_data (ndarray): The ROI data.
        max_iter (int): Maximum number of iterations.
        ct_path (str): The original CT Scan path.

    Returns:
        ndarray: Segmented liver.
    """
```

Input: ct_data, roi_data, max_iter(Optional Variable) (Hard Case 1-5 or Case 1-5)

Output: seeds Liver Segmentation estimations after running MSRG algorithm – uses vectorial\matrixial code in order to run efficiently and dilate the liver segmentation mask according to BFS Algorithm that adds voxels inside the ROI with vectorized code that an homogeneity condition holds for them: Their HU Value is close according to Homogeneity Function as follows:

$$\delta(vox)=\frac{HU\_Value(vox)-MEAN_{y\in R_{seg_{mask}}}\{HU\_Value(y)\}}{STD_{y\in R_{seg_{mask}}}\{HU\_Value(y)\}}$$

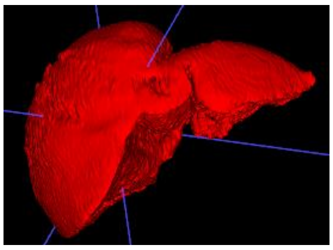
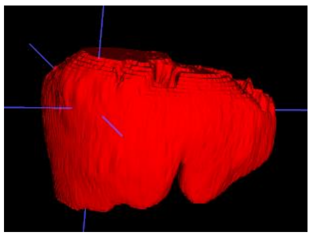
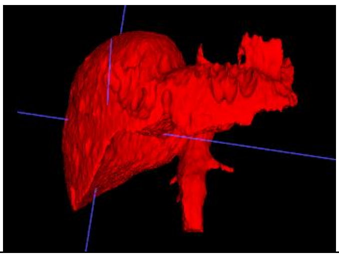
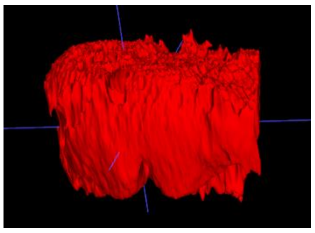
Input Image: **CT Case 1**	Output Image: **CT Case 1 – Liver Segmentation**	Input Image: **CT Case 2**	Output Image: **CT Case 2 – Liver Segmentation**
			

Evaluate Segmentation – Calculate DICE & VOD Scores

Function Name:
```
def evaluate_segmentation(ground_truth_seg,
processed_liver_seg):
    """
    This function evaluates segmentation using DICE and VOD scores.
    :param ground_truth_seg: The ground truth segmentation.
    :param processed_liver_seg: The processed liver segmentation.
    :return: The DICE score and VOD.
    """
    print("Begin Stage 5: Calculate DICE Score & Volume Overlap
Difference ("
          "VOD)")
    # Compute the sum of the volumes
    ground_truth_vol = np.sum(ground_truth_seg)
    processed_liver_vol = np.sum(processed_liver_seg)

    # Compute intersection & union volumes
    intersection_vol = np.sum(
    np.logical_and(ground_truth_seg, processed_liver_seg))
    union_vol = np.sum(np.logical_or(ground_truth_seg,
processed_liver_seg))

    # Compute DICE & VOD Coefficients
    dice = 2.0 * intersection_vol / (ground_truth_vol +
processed_liver_vol)
    vod = 1 - intersection_vol / union_vol

    return dice, vod
```

Input: ground_truth_seg, processed_liver_seg (HardCase 1 or Case 1)

Output: DICE Score, VOD of the comparison between Ground Truth Liver Segmentation & Estimated Calculated Liver Segmentation from MSRG algorithm results.

Input Images:	Outputs	Input Images:	Outputs
Ground Truth File: Case1_liver_segmentation.nii.gz 	$DICE = 0.91$ $VOD = 0.165$	**Ground Truth File:** HardCase1_ liver_segmentation.nii.gz 	$DICE = 0.88$ $VOD = 0.21$
Estimated Segmentation: Case1_post_process _liver_segmentation.nii.gz 		**Estimated Segmentation:** HardCase1_post_process _liver_segmentation.nii.gz 	

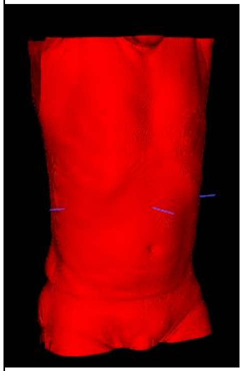
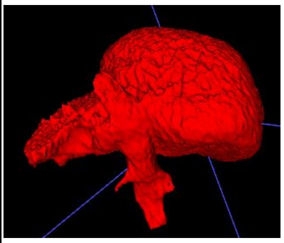
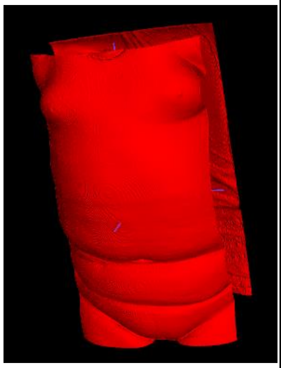
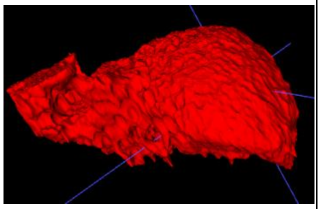
Multiple Seeds Region Growing

Function Name:
```
def segment_liver(ct_filename, aorta_seg_filename,
output_filename):
    """
    This function segments the liver & perform full liver
segmentation end to end including MSRG and Evaluation with DICE & VOD
Scores. Saves the liver segmentation file and for HardCase1 and Case1
outputs also txt file with DICE & VOD Scores of the segmentation.

    :param ct_filename: The filename of the CT scan.
    :param aorta_seg_filename: The filename of the Aorta
segmentation.
    :param output_filename: The filename for the output liver
segmentation.
    """
```

Input: ct_filename, aorta_seg_filename, output_filename (Hard Case 1-5 or Case 1-5)

Output: Returns None but runs full Liver Segmentation with all stages in the process:

1. isolate_body
2. create_liver_roi
3. find_seed + multiple_seeds_RG
4. evaluate_segmentation – Calculate DICE & VOD Scores.

Input Image: **CT Case 1**	Output Image: **CT Case 1 Liver Segmentation**	Input Image: **CT Case 2**	Output Image: **CT Case 2 Liver Segmentation**
			

Helper Functions – for isolate_body and create_liver_roi functions:

```
def get_segmentation_bbox(segmentation):
    """
    Retrieve the bounding box of a segmentation.
    Parameters:
        segmentation (ndarray): The segmentation.
    Returns:
        tuple: The bounding box coordinates in the order
        (x_min, x_max, y_min, y_max, z_min, z_max).
    """
    # Find the minimum and maximum indices for each axis
    x_idx = np.any(segmentation, axis=(1, 2)).nonzero()[0]
    y_idx = np.any(segmentation, axis=(0, 2)).nonzero()[0]
    z_idx = np.any(segmentation, axis=(0, 1)).nonzero()[0]
    x_min, x_max = x_idx[0], x_idx[-1]
    y_min, y_max = y_idx[0], y_idx[-1]
    z_min, z_max = z_idx[0], z_idx[-1]
    # Return the values as six separate variables
    return x_min, x_max, y_min, y_max, z_min, z_max
```

```
def perform_binary_thresholding(ct_data, lower_threshold,
upper_threshold):
    """
    Perform binary thresholding on CT data.

    Args:
        ct_data (ndarray): The CT data to be thresholded.
        lower_threshold (float): The lower threshold value.
        upper_threshold (float): The upper threshold value.

    Returns:
        ndarray: The CT data after thresholding process.
    """
    # Create boolean masks for values above and below the thresholds
    above_lower_threshold = ct_data > lower_threshold
    below_upper_threshold = ct_data < upper_threshold

    # Combine the masks using logical AND
    in_threshold_range = np.logical_and(above_lower_threshold,
                                        below_upper_threshold)
    # Convert boolean mask to integer mask
    scan_post_threshold = in_threshold_range.astype(np.uint8)
    return scan_post_threshold
```

<u>Stage 1 : Isolate Body</u>

```
# ---------------------------------------------------------------------
----------
# Part 2: Liver Segmentation for Hard Cases 1-5 & Regular Cases 1-5


def post_process_bin_ct(binary_img, ball_size=10):
    """
    This function performs post-processing on binary CT data.
    :param binary_img: The binary CT image data.
    :param ball_size: The size of the ball structuring element.
    :param min_size_obj_range: min_size_obj_range: The range of
objects to
    remove.
    :return: The processed binary image.
    """
    footprint = ball(ball_size)
    bin_post_open = binary_opening(binary_img, footprint=footprint)
    bin_post_close = binary_closing(bin_post_open,
footprint=footprint)
    largest_cc = get_largest_component(bin_post_close)
    labeled_scan, num_labels = label(largest_cc,
                                     return_num=True)

    print("Number of Final Connectivity Components:
{}".format(num_labels))
    return largest_cc
```

```
def get_largest_component(mask_scan_data):
    """
    Get the largest connected component from the binary image.
    Args:
        mask_scan_data (ndarray): The binary image.
        min_size_obj_range (tuple): The range of objects to remove.
    Returns:
        ndarray: The largest connected component.
    """
    labeled_scan, num_labels = label(mask_scan_data, return_num=True)
    if num_labels == 1:
        return mask_scan_data
    # Determine the size of each labeled region
    region_sizes = np.bincount(labeled_scan.ravel())

    # Find the largest labeled region index (excluding background
label)
    largest_label = np.argmax(region_sizes[1:]) + 1
    # Create a mask for the largest component
    largest_component_mask = labeled_scan == largest_label
    return largest_component_mask


def isolate_body(ct_data, body_lower_threshold=BODY_LOWER_THRESHOLD,
                 body_upper_thresh=BODY_UPPER_THRESHOLD,
                 ball_size=2, ct_path=''):
    """
    This function isolates the body from CT data and separates
between the
    patient's body mask and the bed of the CT chamber. At the end the
noise
    from the image is cleaned.
    :param ct_data: The CT data.
    :param body_lower_threshold: The lower threshold for the body.
    :param body_upper_thresh: The upper threshold for the body.
    :param ball_size: The size of the ball structuring element.
    :param min_size_obj_range: The range of objects to remove.
    :return: The segmented isolated body.
    """
    print("Begin Stage 1: Isolate Body")
    after_binary_data = perform_binary_thresholding(ct_data,
body_lower_threshold,

body_upper_thresh)
    ct_filename = ct_path.split('/')[-1]
    body_thresholding_path = ct_filename.replace('CT',

'body_thresholding_result')
    save_nifti(after_binary_data, ct_path, f'{MIP_DATA_PATH}/'
                                            f'{SEGMENT_FOLDER}/'

f'{body_thresholding_path}')
    # Perform binary opening and closing operations
    isolated_body_segment = post_process_bin_ct(after_binary_data,
                                                ball_size)
    return isolated_body_segment
def post_process_bin_ct(binary_img, ball_size=10):
    """
    This function performs post-processing on binary CT data.
    :param binary_img: The binary CT image data.
    :param ball_size: The size of the ball structuring element.
    :param min_size_obj_range: min_size_obj_range: The range of
objects to
    remove.
    :return: The processed binary image.
    """
    footprint = ball(ball_size)
    bin_post_open = binary_opening(binary_img, footprint=footprint)
    bin_post_close = binary_closing(bin_post_open,
footprint=footprint)
    largest_cc = get_largest_component(bin_post_close)
    labeled_scan, num_labels = label(largest_cc,
                                      return_num=True)

    print("Number of Final Connectivity Components:
{}".format(num_labels))
    return largest_cc
```

Stage 2 : Create Liver ROI

```
def create_liver_roi(ct_data, aorta_data,
                     liver_lower_th=LIVER_MIN_THRESHOLD,
liver_upper_th=LIVER_MAX_THRESHOLD,
                     liver_roi_x=(-30, 195), liver_roi_y=(-100, 100),
                     liver_roi_z=(-75, 50), ct_path=''):
    """
    This function creates a region of interest (ROI) for the liver.
    :param ct_data: The CT data.
    :param aorta_data: The aorta data.
    :param liver_lower_th: The lower threshold for the liver.
    :param liver_upper_th: The upper threshold for the liver.
    :param liver_roi_x: The X-axis range for the liver ROI.
    :param liver_roi_y: The Y-axis range for the liver ROI.
    :param liver_roi_z: The Z-axis range for the liver ROI.
    :param ct_path: The path to the CT file.
    :return: The liver region of interest.
    """
    # Case 1: (80, 365) (240, 478) (15, 60)
    # Case 2: (38, 215) (95, 345) (52, 186)
    # Case 5: (111, 198) (210, 315) (0, 22)
    body_mask = isolate_body(ct_data, ct_path=ct_path)
    body_mask_for_save = body_mask.astype(np.uint8)
    print("Begin Stage 2: Create Liver ROI")
    ct_filename = ct_path.split('/')[-1]
    isolate_body_res_path = ct_filename.replace('CT',
'isolate_body_result')
    body_without_zeros = np.where(body_mask, ct_data, -np.inf)
    save_nifti(body_mask_for_save, ct_path, f'{MIP_DATA_PATH}/'
                                            f'{SEGMENT_FOLDER}/'

f'{isolate_body_res_path}')
    liver_bin_result =
perform_binary_thresholding(body_without_zeros,
                                                    liver_lower_th,
                                                    liver_upper_th)

    liver_bin_res_path = ct_filename.replace('CT',
'liver_binary_result')
    save_nifti(liver_bin_result, ct_path, f'{MIP_DATA_PATH}/'
                                          f'{SEGMENT_FOLDER}/'
                                          f'{liver_bin_res_path}')
    x_min, x_max, y_min, y_max, z_min, z_max = get_segmentation_bbox(
        aorta_data > 0)
    print(f' Aorta bbox: {(x_min, x_max, y_min, y_max, z_min,
z_max)}')
    # or '4' not in ct_path
    if 'Hard' in ct_path and ('2' not in ct_path):
        liver_roi_x = (-166, 64)
        liver_roi_y = (-26, 124)
        liver_roi_z = (-27, 38)
    if 'Hard' in ct_path and '2' in ct_path:
        liver_roi_x = (-209, -65)
        liver_roi_y = (-110, 92)
        liver_roi_z = (-33, 101)
    if 'Hard' in ct_path and '5' in ct_path:
        liver_roi_x = (-171, 25)
        liver_roi_y = (-71, 93)
        liver_roi_z = (-12, 10)
    liver_x_min, liver_x_max = np.maximum(0,
                                          x_min + liver_roi_x[0]),
np.minimum(
        aorta_data.shape[0], x_max + liver_roi_x[1])
    liver_y_min, liver_y_max = np.maximum(0,
                                          y_min + liver_roi_y[0]),
np.minimum(
        aorta_data.shape[1], y_max + liver_roi_y[1])
    liver_z_min, liver_z_max = np.maximum(0, (
            (z_min + z_max) // 2) + liver_roi_z[0]), np.minimum(
        aorta_data.shape[2], ((z_min + z_max) // 2) + liver_roi_z[1])
    liver_mask = np.zeros_like(liver_bin_result)
    liver_mask[liver_x_min:liver_x_max, liver_y_min:liver_y_max,
    liver_z_min:liver_z_max] = 1
    liver_roi = np.logical_and(liver_mask, liver_bin_result)
    liver_roi_path = ct_filename.replace('CT', 'liver_roi_result')
    save_nifti(liver_roi.astype(np.uint8), ct_path,
f'{MIP_DATA_PATH}/'
                                          f'{SEGMENT_FOLDER}/'
                                          f'{liver_roi_path}')
    return liver_roi
```

$\underline{\textit{Stage 3A}: \ \textit{Find Suitable in} - \textit{region Seeds}}$

```
def get_seeds_inside_liver_roi(liver_roi, liver_roi_x=(-30, 180),
                               liver_roi_y=(-100, 100)):
    """
    This function retrieves seeds inside the liver ROI.
    :param liver_roi: The liver region of interest.
    :param liver_roi_x: The X-axis range for the liver ROI.
    :param liver_roi_y: The Y-axis range for the liver ROI.
    :return: The seeds inside the liver ROI.
    """
    x_min, x_max, y_min, y_max, z_min, z_max =
get_segmentation_bbox(liver_roi)
    roi_min_x, roi_max_x = int(x_max - 0.7 * liver_roi_x[1]),
int(x_max - 0.2
                                                               *
liver_roi_x[1])
    avg_y = (y_max + y_min) // 1.75
    roi_min_y, roi_max_y = int(avg_y - liver_roi_y[1] / 3), int(avg_y
+
liver_roi_y[1] / 3)
    avg_z = (z_max + z_min) // 2
    roi_min_z, roi_max_z = int(avg_z - 8), int(avg_z + 8)
    seeds_in_roi = np.zeros_like(liver_roi).astype(bool)
    seeds_in_roi[roi_min_x:roi_max_x, roi_min_y:roi_max_y,
    roi_min_z:roi_max_z] = True
    return seeds_in_roi


def get_seeds_lowest_gradients_sorted(roi_data, seeds):
    """
    This function sorts seeds based on the lowest gradient
magnitudes.
    :param roi_data: The region of interest data.
    :param seeds: The seed points.
    :return: The sorted seeds.
    """
    # Compute the gradient components
    dx = np.gradient(roi_data, axis=0)
    dy = np.gradient(roi_data, axis=1)
    dz = np.gradient(roi_data, axis=2)
    # Compute the squared gradient magnitudes
    gradients_magnitudes = dx ** 2 + dy ** 2 + dz ** 2
    # Flatten the gradient image and select values corresponding to
seed points
    seed_grads =
gradients_magnitudes.flatten()[np.ravel_multi_index(seeds.T,

gradients_magnitudes.shape)]
    # Sort the indices based on gradient magnitudes
    seeds_order = np.lexsort((range(len(seed_grads)), seed_grads))
    # Reorder the seeds based on the sorted indices
    sorted_seeds = seeds[seeds_order]
    return sorted_seeds


def normalize_seeds_hu_values(ct_data, seeds):
    """
    This function calculates the mean of the seed all_seeds_neighbors
and assigns it
    to the seed HU Value in the CT Scan.
    :param ct_data: The CT data.
    :param seeds: The seed points.
    :return: CT Scan with updated seeds values.
    """
    # Get the all_seeds_neighbors for all seeds at once
    all_seeds_neighbors = np.array([get_single_voxel_neighbors(seed)
                                    for seed in seeds])
    # Calculate the mean for each seed's all_seeds_neighbors and
update
    # the corresponding voxel in ct_data
    for seed, seed_neighbors in zip(seeds, all_seeds_neighbors):
        ct_data[tuple(seed)] =
np.mean(ct_data[tuple(seed_neighbors.T)])

    return ct_data



def sample_random_points(roi_data, num_seeds):
```

```python
    """
    This function samples random points from the ROI.
    :param roi_data: The region of interest data.
    :param num_seeds: The number of seeds to sample.
    :return: The sampled points.
    """
    # Find the indices of the ROI points
    roi_points_indices = np.transpose(np.where(roi_data == 1))
    # Sample random indices without replacement
    random_indices = random.sample(range(len(roi_points_indices)),
num_seeds)
    # Extract the sampled points from the ROI points array
    sampled_points = [tuple(roi_points_indices[idx]) for idx in
random_indices]
    return sampled_points




def find_seeds(roi_data, num_seeds=NUM_SEEDS):
    """
    This function finds seeds for region growing.
    :param roi_data: The region of interest data.
    :param num_seeds: The number of seeds to find.
    :return: The seeds.
    """
    # Sample random points from the ROI
    seeds_list = sample_random_points(roi_data, num_seeds)
    return seeds_list
```

Stage 3B : run MSRG Algorithm

```python
def get_all_next_front_voxels(voxels_coords):
    """
   Get all neighboring voxels of given coordinates.

   Args:
       voxels_coords (ndarray): The coordinates of the voxels.

   Returns:
       ndarray: All neighboring voxels.
   """
    # Define the neighbors_idx_mask for 26 all_next_front_voxels
    neighbors_idx_mask = np.array([
        [-1, -1, -1], [-1, -1, 0], [-1, -1, 1],
        [-1, 0, -1], [-1, 0, 0], [-1, 0, 1],
        [-1, 1, -1], [-1, 1, 0], [-1, 1, 1],
        [0, -1, -1], [0, -1, 0], [0, -1, 1],
        [0, 0, -1], [0, 0, 1],
        [0, 1, -1], [0, 1, 0], [0, 1, 1],
        [1, -1, -1], [1, -1, 0], [1, -1, 1],
        [1, 0, -1], [1, 0, 0], [1, 0, 1],
        [1, 1, -1], [1, 1, 0], [1, 1, 1]
    ])
    # Add neighbors_idx_mask to each point
    all_next_front_voxels = voxels_coords[:, None, :] +
neighbors_idx_mask
    # Exclude the original points
    return np.unique(all_next_front_voxels.reshape(-1, 3), axis=0)
```

```
def get_single_voxel_neighbors(voxel_coords):
    """
    This function retrieves all neighbors of given voxel indices at
once.
    :param voxel_coords: The coordinates of the voxels.
    :return: The neighbors of the voxels.
    """
    # Define voxels_idx_mask for 26 neighbors in 3D excluding the
voxel itself
    voxels_idx_coords = np.array(np.meshgrid([-1, 0, 1], [-1, 0, 1],
[-1, 0,

1])).T.reshape(-1, 3)
    # Compute neighbor coordinates by adding voxels_idx_mask to the
voxel_coord
    neighbors = np.array(voxel_coords) + voxels_idx_coords
    # Exclude the voxel itself
    neighbors = neighbors[np.any(voxels_idx_coords != 0, axis=1)]
    return neighbors


def homogeneity_condition(ct_data, segmented_liver, voxels_to_check,
                          threshold=0.9):
    """
    Check homogeneity condition for voxels.
    Args:
        ct_data (ndarray): The CT data.
        segmented_liver (ndarray): The segmented liver data.
        voxels_to_check (ndarray): The voxels to check.
        threshold (float): The threshold value for homogeneity.

    Returns:
        ndarray: Boolean mask indicating homogeneity condition
satisfaction.
    """
    regions_hu = ct_data[segmented_liver]
    voxels_to_add_hu = ct_data[voxels_to_check]
    regions_mean = np.mean(regions_hu)
    regions_std = np.std(regions_hu)
    return np.abs((regions_mean - voxels_to_add_hu) /
                  np.maximum(1, regions_std)) <= threshold


def multiple_seeds_RG(ct_data, roi_data, max_iter=MAX_ITERS,
ct_path=''):
    """
    Perform Multi-Seeded Region Growing algorithm fully with all
parts.

    Args:
        ct_data (ndarray): The CT data.
        roi_data (ndarray): The ROI data.
        max_iter (int): Maximum number of iterations.
        ct_path (str): The original CT Scan path.

    Returns:
        ndarray: Segmented liver.
    """
    print("Begin Stage 3: Find Seeds with lowest Gradient Values")
    seeds_roi = get_seeds_inside_liver_roi(roi_data)
    seeds = np.array(find_seeds(seeds_roi))
    grad_sorted_seeds = get_seeds_lowest_gradients_sorted(ct_data,
seeds)
    print("Done finding Seeds...")
    print("Begin Stage 4: Run MSRG Algorithm")
    liver_segmentation = run_msrg_algorithm(ct_data, roi_data, seeds,
                                            grad_sorted_seeds,
max_iter,
                                         ct_path)
    print("Done performing Multi Seeded Region Growing...")
    return liver_segmentation


def run_msrg_algorithm(ct_data, roi_data, seeds, grad_sorted_seeds,
max_iter,
                       ct_path):
    """
    Run Multi-Seeded Region Growing algorithm main logics.

    Args:
        ct_data (ndarray): The CT data.
        roi_data (ndarray): The ROI data.
        seeds (ndarray): Seed points.
        grad_sorted_seeds (ndarray): Seed points ordered by lowest
gradient.
        max_iter (int): Maximum number of iterations.
        ct_path (str): The CT path.
    Returns:
        ndarray: Segmented liver.
    """
    global MIN_QUEUE_SIZE
    ct_data = normalize_seeds_hu_values(ct_data, seeds)
    visited_matrix = np.zeros_like(ct_data).astype(bool)
    liver_segmentation = np.zeros_like(ct_data).astype(bool)
    liver_segmentation[tuple(grad_sorted_seeds.T)] = True
    ct_filename = ct_path.split('/')[-1]
    seeds_mask_path = ct_filename.replace('CT', 'seeds_mask_result')
    save_nifti(liver_segmentation.astype(np.uint8), ct_path,
f'{MIP_DATA_PATH}/'
                                          f'{SEGMENT_FOLDER}/'
                                          f'{seeds_mask_path}')
    visited_matrix[tuple(grad_sorted_seeds.T)] = True
    if 'Hard' in ct_path:
        max_iter = HARD_CASE_ITERS
        MIN_QUEUE_SIZE = 0
    front_voxels = get_all_next_front_voxels(grad_sorted_seeds)
    queue = np.array(front_voxels)

    iter_counter = 0
    while iter_counter < max_iter and queue.shape[0] >
MIN_QUEUE_SIZE:
        valid_voxels, homogeneity_mask =
get_valid_coords_and_homogeneity_mask(
            ct_data, liver_segmentation, queue, visited_matrix,
roi_data, ct_path)
        if not np.any(valid_voxels):
            break
        iter_counter, queue = dilate_liver_seg_mask(homogeneity_mask,
                                                    iter_counter,

liver_segmentation,
                                                            valid_voxels,
visited_matrix)

    liver_segmentation =
post_process_liver_segmentation(liver_segmentation)
    return liver_segmentation


def dilate_liver_seg_mask(homogeneity_mask, iter_counter,
                          segmented_liver, valid_voxels,
visited_matrix):
    """
    Dilate liver segmentation mask.
    Args:
        homogeneity_mask (ndarray): Homogeneity mask.
        iter_counter (int): Iteration counter.
        segmented_liver (ndarray): Segmented liver data.
        valid_voxels (ndarray): Valid coordinates.
        visited_matrix (ndarray): Visited points.
    Returns:
        tuple: Updated iteration counter and queue.
    """
    added_voxels_mask = valid_voxels[homogeneity_mask]
    segmented_liver[tuple(added_voxels_mask.T)] = True
    visited_matrix[tuple(valid_voxels.T)] = True
    queue = get_all_next_front_voxels(added_voxels_mask)
    iter_counter += 1
    print(queue.shape[0])
    print(f"Iteration {iter_counter} Done")
    return iter_counter, queue


def get_valid_coords_and_homogeneity_mask(ct_data,
liver_segmentation, queue,
                                          visited_matrix, roi_data,
ct_path):
    """
    Get valid coordinates and homogeneity mask.

    Args:
        ct_data (ndarray): The CT data.
        liver_segmentation (ndarray): The segmented liver data.
        queue (ndarray): Queue of points.
        visited_matrix (ndarray): Visited points.
        roi_data (ndarray): The ROI data.
        ct_path (str): The CT path.

    Returns:
        tuple: Valid coordinates and homogeneity mask.
    """
    roi_voxel_seg = are_voxels_inside_roi(queue.T,
*get_segmentation_bbox(roi_data))
    valid_voxels = queue[roi_voxel_seg]
    threshold = 0.9 if 'Hard' in ct_path else 1.2
    homogeneity_matrix = homogeneity_condition(ct_data,
liver_segmentation,
                                               tuple(valid_voxels.T),
                                               threshold=threshold)
    unvisited_matrix = homogeneity_matrix &
~visited_matrix[tuple(valid_voxels.T)]
    return valid_voxels, homogeneity_matrix & unvisited_matrix
```

```python

def post_process_liver_segmentation(liver_segmentation):
    """
    Post-process liver segmentation.
    Args:
        liver_segmentation (ndarray): The segmented liver data.

    Returns:
        ndarray: Processed segmented liver data.
    """
    foot_print = ball(BALL_RADIUS)
    liver_segmentation = binary_closing(liver_segmentation,
footprint=foot_print)
    return liver_segmentation


def are_voxels_inside_roi(voxels_to_verify, x_min, x_max, y_min,
y_max, z_min, z_max):
    """
    This function checks if voxels are inside the ROI.
    :param voxels_to_verify: The coordinates of the points.
    :param x_min: The minimum X-axis value of the ROI.
    :param x_max: The maximum X-axis value of the ROI.
    :param y_min: The minimum Y-axis value of the ROI.
    :param y_max: The maximum Y-axis value of the ROI.
    :param z_min: The minimum Z-axis value of the ROI.
    :param z_max: The maximum Z-axis value of the ROI.
    :return: Boolean array indicating if points are inside the ROI.
    """
    # Check if voxels are inside the ROI vectorized operations
    x_in_range = np.logical_and(voxels_to_verify[0, ] >= x_min,
                                voxels_to_verify[0] <= x_max)
    y_in_range = np.logical_and(voxels_to_verify[1, ] >= y_min,
                                voxels_to_verify[1] <= y_max)
    z_in_range = np.logical_and(voxels_to_verify[2, ] >= z_min,
                                voxels_to_verify[2] <= z_max)
    return np.logical_and(np.logical_and(x_in_range, y_in_range),
                          z_in_range).flatten()
```

<u>Stage 4 : Evaluate Segmentation VOD & DICE Scores</u>

```python
        def evaluate_segmentation(ground_truth_seg,
processed_liver_seg):
    """
    This function evaluates segmentation using DICE and VOD scores.
    :param ground_truth_seg: The ground truth segmentation.
    :param processed_liver_seg: The processed liver segmentation.
    :return: The DICE score and VOD.
    """
    print("Begin Stage 5: Calculate DICE Score & Volume Overlap
Difference ("
          "VOD)")
    # Compute the sum of the volumes
    ground_truth_vol = np.sum(ground_truth_seg)
    processed_liver_vol = np.sum(processed_liver_seg)

    # Compute intersection & union volumes
    intersection_vol = np.sum(
    np.logical_and(ground_truth_seg, processed_liver_seg))
    union_vol = np.sum(np.logical_or(ground_truth_seg,
processed_liver_seg))

    # Compute DICE & VOD Coefficients
    dice = 2.0 * intersection_vol / (ground_truth_vol +
processed_liver_vol)
    vod = 1 - intersection_vol / union_vol

    return dice, vod
```

Stage 5 : Segment Liver – Run Full Segmentation Algorithm

```
def segment_liver(ct_filename, aorta_seg_filename, output_filename):
    """
    This function segments the liver & perform full liver
segmentation end
     to end including MSRG and Evaluation with DICE & VOD Scores.
     Saves the liver segmentation file and for HardCase1 and Case1
     outputs also txt file with DICE & VOD Scores of the
segmentation.
    :param ct_filename: The filename of the CT scan.
    :param aorta_seg_filename: The filename of the Aorta
segmentation.
    :param output_filename: The filename for the output liver
segmentation.
    """
    ct_data, ct_filename, ct_nifti_file =
load_nifti(f'{MIP_DATA_PATH}'
                                                 f'/{ct_filename}')
    aorta_data, aorta_seg_filename, _ = load_nifti(f'{MIP_DATA_PATH}'

f'/{aorta_seg_filename}')

    liver_seg_output_path =
f'{MIP_DATA_PATH}/{SEGMENT_FOLDER}/{output_filename}'
    seg_folder_path = f'{MIP_DATA_PATH}/{SEGMENT_FOLDER}'
    create_folder(seg_folder_path)
    liver_roi = create_liver_roi(ct_data, aorta_data,
ct_path=ct_filename)
    liver_segmentation = multiple_seeds_RG(ct_data, liver_roi,
ct_path=ct_filename)
    save_nifti(liver_segmentation.astype(np.uint8), ct_filename,
liver_seg_output_path)

    liver_ground_truth_path = ct_filename.replace('CT',
'liver_segmentation')

    if os.path.exists(liver_ground_truth_path):
        gt_liver_seg, gt_path, _ =
load_nifti(liver_ground_truth_path)
        dice, vod = evaluate_segmentation(gt_liver_seg,
liver_segmentation)
        print(f"DICE Score is: {dice}\nVOD Score is: {vod}")
        full_output_path = liver_seg_output_path
        with open(full_output_path.replace('.nii.gz', '.txt'), 'w+')
as f:
            f.write(f" Ground Truth File: {liver_ground_truth_path}\n
VS "
                    f"\nCalculated "
                    f"Liver Segmentation: {liver_seg_output_path}\n"
                    f" DICE Score & VOD Score\n")
            f.write(f"DICE is: {dice}\nVOD is: {vod}\n")
```