Lung Vessel and Fissure Segmentation for CT Scans

Hanxiang Hao (hh595)

Advisor: Prof. Anthony P. Reeves

Contents

Abstract

Segmentation of lung vessel and fissure is essential and significant for clinical practice, especially for lung lobes segmentation, which is also challenging for the cases with pulmonary diseases. In this project, I implement the algorithm based on Lassen et al. [1] to segment the lung vessel and fissure for computed tomography (CT) scans. The vessel segmentation is based on 3D connected component analysis and morphological operation, while the fissure segmentation is based on Hessian matrix analysis and vector-based region growing. The dataset I used is the LOLA11 which contains 55 cases and LR SIMBA LIDC Image Dataset which contains 1000 cases. The final accuracy for the fissure segmentation is 6.23mm for the average distance between each voxel in the manually marked reference and the closest voxel in the fissure segmentation for LOLA11 dataset.

Key words: CT scans, segmentation, lung vessel, lung fissure.

1. Introduction

Lung lobe segmentation is relevant in clinical applications particularly for treatment planning, since the location and distribution of pulmonary diseases are important parameters for the selection of a suitable treatment [1]. Furthermore, lung vessel and lung fissure are important features for lung lobes segmentation. Since there are usually no major vessels at the lobar boundaries, the distance to the pulmonary vasculature is a suitable feature to detect lobar boundary. Moreover, fissures can locally be modeled as a sheet; therefore, we can use Hessian matrix to extract the feature for fissure.

1.1 Background

The human lungs are subdivided into five lobes that are separated by visceral pleura called pulmonary fissure. There are three lobes in the right lung, namely upper, middle, and lower lobe. The right upper and right middle lobe are divided by the right minor fissure whereas the right major fissure delimits the lower lobe from the rest of the lung. In the left lung there are only two lobes, the upper and the lower lobe, that are divided by the left major fissure. The lobar fissures are low contrast surfaces with blurred boundaries when viewed on cross-sectional CT images. Computer based detection of the fissures is complicated by surrounding vessels and other structures, and noise and artifacts in the images. The example of lung CT scans is shown in the figure 1.

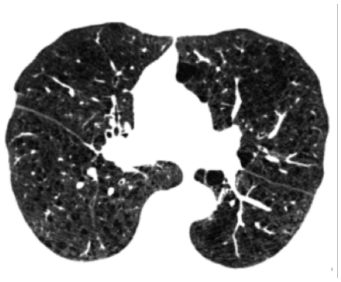

Figure 1 The original CT scans with lung mask

1.2 Previous Work

Table 1 Comparison of Previous Works

	Methods	Dataset CT Scans	Segmentation	Accuracy
1	Rikxoort et al. 2009 [4]	100 low-dose	• Region growing for lung segmentation Superivsed filter for fissure segmentation Voxel classification for lobes segmentation	77%
2	Pu et al. 2009 [5]	65	• Detecting plane patches in sub-volumes in the lungs Using Radial Basis Functions represent fissures	50.8% (rates as "excellent" or "good" by two radiologists)
3	Lassen et al. 2013 [1]	75	• Vessel segementation with region growing Fissure enhancement by Hessian Matrix Bronchi segmentation by enhancement filter Watersheld based on vessel, fissure and bronchi	88%
4	Li et al. 2006 [2]	38	• Fuzzy resoning system to search fissure based on the following three sources: image intensities, an anatomic smoothness constraint, and the atlas-based search initialization	99.8%
5	Wiemker et al. 2014 [3] 2005	-	• This paper suggests two possible 3D filter approaches: 1. Filter is based on first derivatives of the image gray values and utilizes the eigenvalues of the local structure tensor; 2. Filter is based on second derivatives and utilizes the eigenvalues of the local Hesse matrix.	-

"-" means the data is not provided.

1.3 Report Outline

We are going to discuss the main algorithm of our project, the database and the performance metrics we used to test our method in part 2, and the results will be included in part 3 and discussions will be stated in part 4.

2. Methods

In this section, we will discuss the main algorithm of our project, the performance metrics we use to test our method, and the database we plan to use.

2.1 Lung Vessel and Lung Fissure Segmentation

This project mainly contains two parts: lung vessel segmentation and lung fissure extraction. The vessel segmentation is based on 3D connected component analysis and morphological operation, while the fissure segmentation is based on Hessian matrix analysis and vector-

based region growing. In the first step, lungs are segmented since all other segmentations are only performed inside the lung regions.

2.1.1 Lung segmentation

A good lung segmentation is significant for the lobes segmentation, since each sub-step is based of the original image with the lung mask. Generally, the method includes two steps: region growing and morphological operation. First, the algorithm starts with the threshold-based 3D region growing for pulmonary airspace segmentation and the two seed points from left and right lung region are chosen manually. By implementing the region growing, a rough area of lung area is obtained. Furthermore, a morphological closing step is performed to close major interior holes resulting in the previous step. The example of the original image and the lung mask is shown in the figure 2.

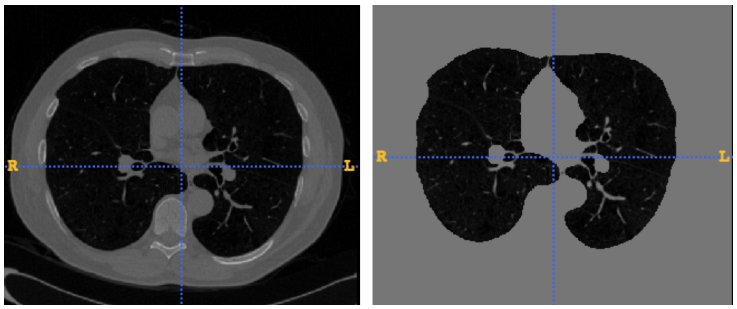

Figure 2 The original image (left) and the lung mask (right)

2.1.2 Pulmonary Vessel Enhancement

According to Lassen et al. [1] there is no major vessel at the lobes boundaries; therefore, it is reasonable and useful to use the distance to the vasculature as a feature to detect the boundaries of lobes.

The first step of the vessel enhancement is a downscaling of the original image for reducing the memory requirement, since the intensity difference between vessels and background is significant. The downscaling procedure is defined as a following equation:

$$v_{ds} = \begin{cases} \max\left(0, \min\left(254, \frac{v_{original} + 1024}{4}\right)\right), v \in L \\ 255, otherwise \end{cases}$$

where the v_{ds} and $v_{original}$ are the voxel value of downscaling and original image and L is the set of voxel in the foreground of lung mask image. Since the range of the voxel in the original image is from [-1024, 1024], the downscaling image ranges from [0, 255].

After downscaling, a fixed-value thresholding operation is implement to the downscaling image. A fixed threshold of 130 was empirically set with a consideration of the tradeoff between sensitivity and specificity. Furthermore, the connected component analysis is implemented and afterwards, we remove the region with the volume less than 2ml (668 voxels for the LOLA dataset). Finally, we use morphological closing with a 3-voxel-radius sphere kernel to fill the gaps between each vessel. The result of vessel enhancement is shown in the figure 3 and figure 4.

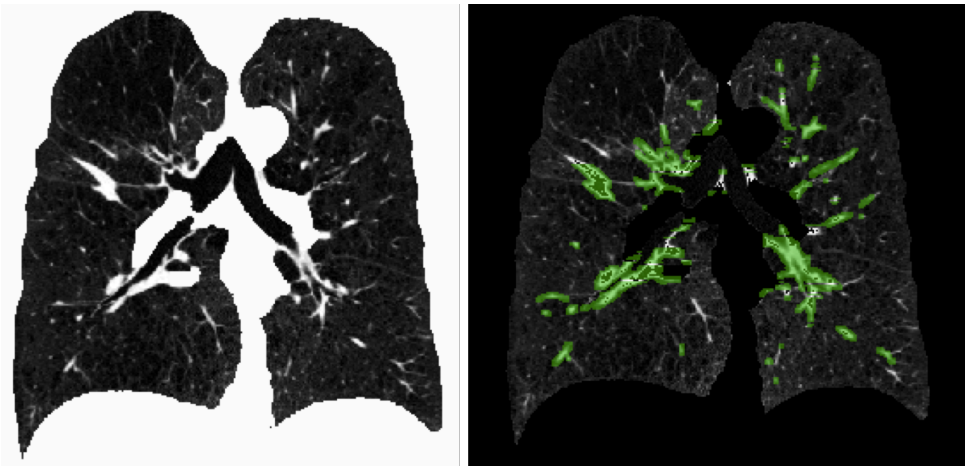

Figure 3 The lung mask (left) and vessel enhancement (right)

Figure 4 Lung Vessel in 3D view

2.1.3 Pulmonary Fissure Segmentation

The method of fissure extraction is based on the eigenvalues of the Hessian matrix that gives a fissure probability for each voxel in lung mask. The lung mask image is generated based on the lung region obtained in the first step and the original 16-bit image rather than 8-bit downscaling image. Furthermore, the definition of Hessian matrix shows below:

$$H = \begin{bmatrix} g_{xx} & g_{xy} & g_{xz} \\ g_{yx} & g_{yy} & g_{yz} \\ g_{zx} & g_{zy} & g_{zz} \end{bmatrix}$$

where g_{**} is the second derivative of Gaussian. This approach uses the fact that, at a planar structure like a fissure, there exists a strong curvature of the gray value profile perpendicular to the fissure and two vanishing curvatures parallel to the fissure. From the matrix above, the three eigenvalues λ_0, λ_1, λ_2, where $|\lambda_0| \geq |\lambda_1| \geq |\lambda_2|$ can be obtained. Fissures can locally be modeled as a sheet where the eigenvalue orthogonal to the fissure plane is large, and the other two eigenvalues are small. [1] Thus, on the bright fissures, the ideal relationship is defined as

$$|\lambda_1| = |\lambda_2| = 0 \text{ and } \lambda_0 << 0$$

According to Lassen et al. [1], we then define two features to measure the characteristics of fissure.

$$F_{structure} = \Theta(-\lambda_0) e^{-(\lambda_0 - \alpha)^6 / \beta^6}$$

$$F_{sheet} = e^{-{\lambda_1}^6 / \gamma^6}$$

$F_{structure}$ rates the strength of image structure. To estimate suitable values for α and β, we analyzed $|\lambda_0|$ values of from the datasets with given fissure and vessel mask. The analysis revealed that fissure voxels show $|\lambda_0|$ values between 100 and 250. Vessels show much higher $|\lambda_0|$ values but the $|\lambda_0|$ values of small vessels can go down to around 60. Since we prefer sensitivity over specificity we choose α=160 and β=120 for the distribution function of $F_{structure}$. The term $\Theta(-\lambda_0)$ describes a Heaviside function that sets to 0 for voxels with $\lambda_0 > 0$, i.e., a dark structure on a bright background is not a fissure. The figure 5 shows the graph of the two features.

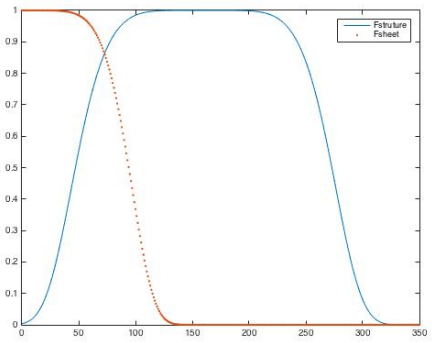

Figure 5 The graph for Fstructure and Fsheet given different $|\lambda_0|$ and $|\lambda_1|$

The F_{sheet} feature discriminates between a sheet structure and other structures such as nodules or vessels, as these latter structures have larger $|\lambda_1|$ values. γ is empirically set to 100 by investigating typical values for fissures. $F_{structure}$ and are in the range [0, 1]. The two features are combined to the overall fissure similarity measure $S_{fissure}$

$$S_{fissure} = F_{structure}F_{sheet}$$

The result of the fissure enhancement is for each voxel a fissure similarity value between 0 and 1.

Furthermore, a mask C is constructed which describes all candidate fissure voxels that is obtained by thresholding of the voxel with too small $S_{fissure}$ value. We empirically set it to 0.8.

$$C = S_{fissure} > 0.8 \cap I < \mu_{vessel} - 2\sigma_{vessel}$$

where I is the intensity of voxel and μ_{vessel} and σ_{vessel} are the mean and standard deviation of vessel we get in the previous step, since the fissure has less intensity than the vessel.

The final step of fissure enhancement is using vector based region growing to remove the noise in the previous step. Since the direction of the eigenvector corresponding with $|\lambda_0|$ is perpendicular with the direction of the fissure surface, we can use the vector to implement a vector-based region growing to extract these voxels by comparing the inner product of the eigenvector of the voxel with its 6 neighbors. We empirically set the thresholding value of region growing to 0.9 since the inner product can be less than 1 due to noise. All 3D components with a volume of at least 0.1 ml are kept to obtain all significant fissure parts and remove most of the noise. Afterwards, a morphological closing with a cubic kernel of 7*7*7 voxels is applied to close minor gaps. This result of final fissure extraction is shown in figure 6 and figure 7.

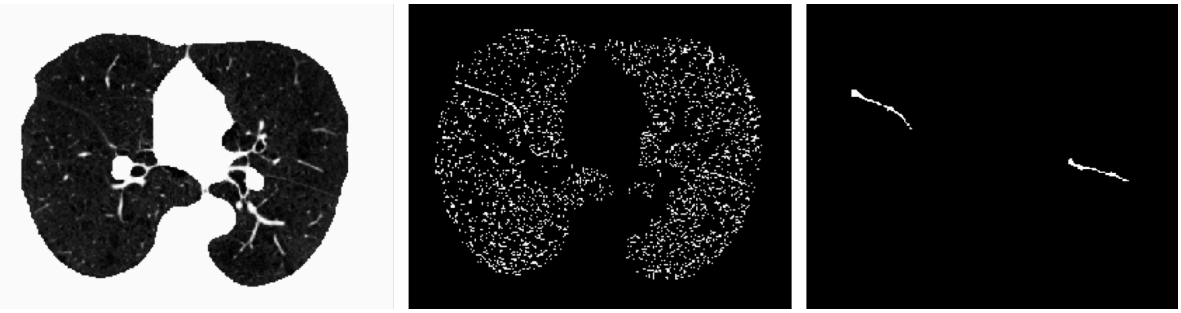

Figure 6 Axial view of the fissure segmentation for LOLA11 case 1

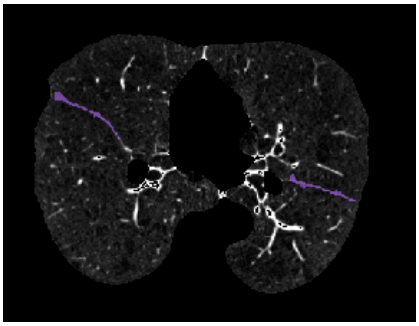

Figure 7 The result of vessel and fissure segmentation

(vessel is labeled as blue and fissure is labeled as green)

2.2 Experiment

This section discusses the selection of parameters of the proposed algorithms and the result of vessel and fissure extraction.

2.2.1 Hypothesis and evaluation function

Per Lassen et al. [1], they applied a watershed-based algorithm to segment lung lobes, and the validation results showed that the mean distance for the segmentation at 0.86mm. Therefore, in our project, we expect that we can get a relatively high accuracy. But since we did not consider the bronchi and vessel when we segment the fissure, we can only extract the visible fissures rather than the whole fissure structure (there are some parts of fissure is invisible due to the lack of thickness), we cut down our hypothetic accuracy from 5.0mm to 10.0mm.

In this project, we use the mean distance from the manually drawn reference as the performance metrics for each lobar border in 3-D by computing the distance between each voxel in the reference standard and the closest voxel in the lobar segmentation.

2.2.2 Documented Data Set

In this project, we use the LOLA11 dataset which consists of 55 cases including both normal and abnormal cases and LR SIMBA LIDC Image Dataset which contains 1000 cases. The organizers of LOLA11 have available a manual segmentation of the lung lobes on nine coronal slices for each case by two human observers. Both observers were instructed not to draw a lobar border when they felt it was not possible. The scans come from a variety of sources and represent a variety of clinically common scanners and protocols. The scans have been selected such that in approximately half of the scans lung and/or lobe segmentation is deemed 'easy' and in the other half 'hard. The maximum slice spacing present is 1.5mm, where most scans are (near) isotropic.

2.2.3 Experiment procedure

In this project, we use one image CT scans for training. The reference standard fissure segmentation is manually segmented by myself. For the vessel segmentation, there are 3 parameters that are critical for the accuracy of segmentation, the fixed threshold value (T_{fixed}), the threshold value for 3D connected component analysis (T_c) and the kernel size for morphological closing ($K_{closing}$). For fissure extraction, there are 6 parameters that we need to explore, the 4 parameters for feature representation (α, β, γ, σ), the threshold value for mask C generation (T_{mask}) and the threshold value for 3D vector-based region growing (T_{rg}).

3. Results

The parameters we got from the experiment are shown as the form below.

Table 2 Parameters for Vessel Segmentation for both LOLA11 and LR dataset

T_{fixed}	T_c	$K_{closing}$
160	0.5	7

T_{fixed}: The fixed threshold value;
T_c: The threshold value for 3D connected component analysis;
$K_{closing}$: Kernel size for morphological closing.

Table 3 Parameters for Fissure Segmentation for LOLA11 dataset

α	β	γ	σ	T_{mask}	T_{rg}
60	120	100	0.5	0.8	0.9

α, β, γ: The parameters for fissure structure representation;
σ: The standard deviation for 2ed-Derivative of Gaussian function in Hessian matrix;
Tmask:The threshold value for mask generation;
Trg: The threshold value for 3D region growing.

Table 4 Parameters for Fissure Segmentation for LR dataset

α	β	γ	σ	T_{mask}	T_{rg}
90	150	120	0.5	0.8	0.9

α, β, γ: The parameters for fissure structure representation;
σ: The standard deviation for 2ed-Derivative of Gaussian function in Hessian matrix;
Tmask:The threshold value for mask generation;
Trg: The threshold value for 3D region growing.

To evaluate the segmentation result of fissure, I manually marked a reference ground truth, which is shown in figure 8.

Figure 8 Ground Truth for Fissure Segmentation for the case 1 in LOLA11 (axial, sagittal and coronal view from left to right)

Table 3 shows the results of the fissure segmentation with a direct comparison to the results of the methods by van Rikxoort et al. [6] and Kuhnigk et al. [7] and the preliminary approach of the proposed algorithm presented by Lassen et al. [1].

Table 5 Fissure Segmentation Accuracies

Fissure Segmentation	Average distance from reference (mm)
The proposed algorithm	6.23
Lassen [1]	0.86
Rikxoort [6]	1.00
Kuhinigk [7]	2.78

The 3D view of the vessel and fissure results are shown in figure 9 and 10.

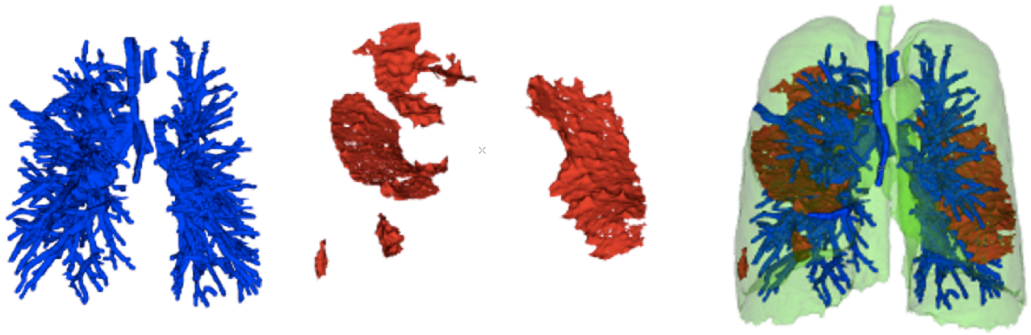

Figure 9 The results of vessel (left), fissure (middle) and combination with lung mask (right) for the case 1 in LOLA 11

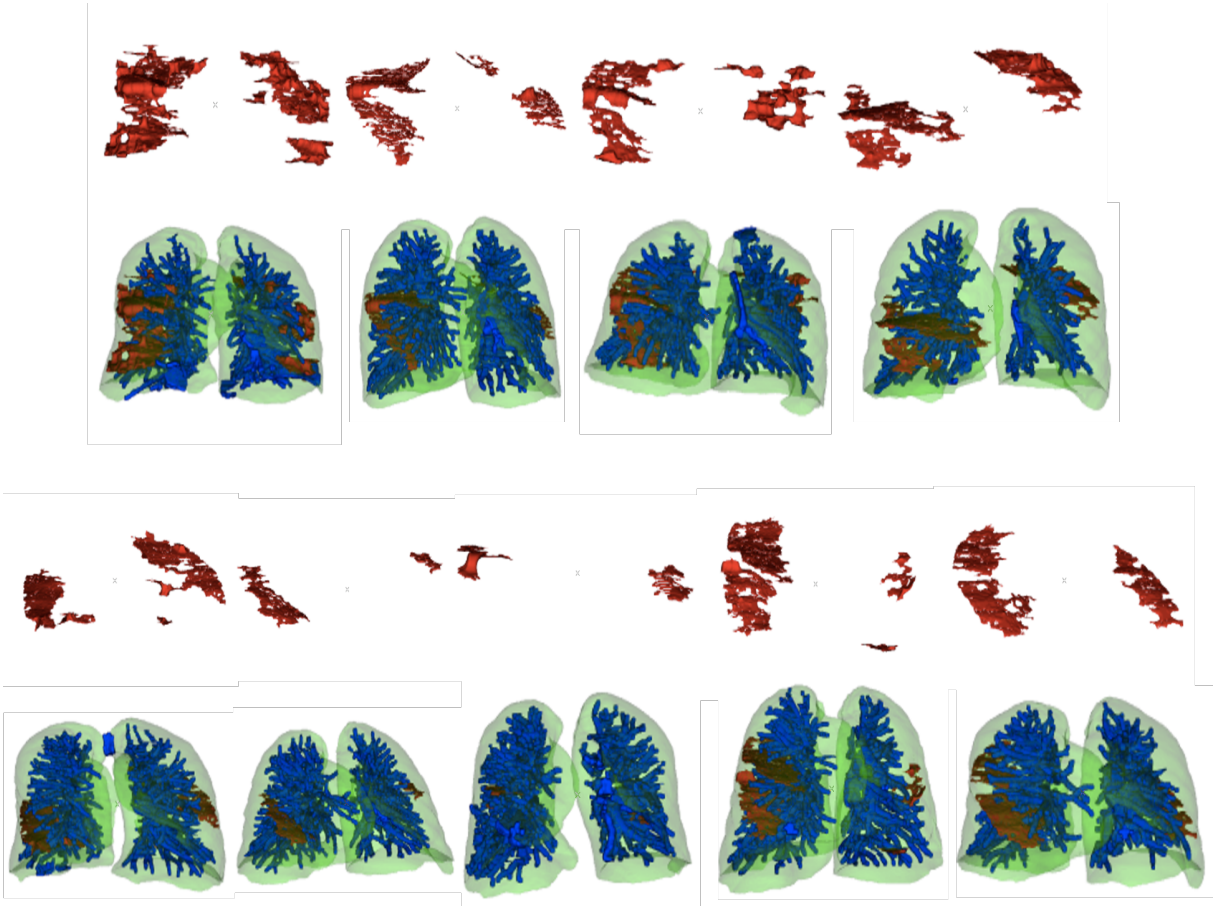

Figure 10 Results for LR Dataset (red region is fissure; blue region is vessel and green region is lung)
The first row is the results of the cases: LC0002, LC0009, LC0015, LC0026;
The Second row is the results of the cases: LC0034, LC0036, LC0039, LC0057.

The LR dataset is challenging for the proposed algorithm, because the sizes of voxel for different images are different. Since the voxel size determines the width of the fissure and vessel, we need to adjust the parameters in the fissure structure representation as mention in the section 2.1.3 to differentiate these two structures, which means if the difference of voxel size between images is overlarge, we need to manually tune the parameters for different images; therefore, the proposed algorithm can automatically segment fissures only if the difference of voxel size within the dataset is small.

4. Discussion

This section discusses the main findings in our project, and furthermore, we also give special case that our algorithm cannot deal with.

4.1 Discussion of the Lung Segmentation

The identification of the lungs in thoracic computed tomography data is a prerequisite for the proposed algorithm. Due to the high-density difference between the air-filled lung regions and soft tissue in CT images, the pulmonary airspace can be identified without greater efforts. However, despite the high contrast of the air-filled lung parenchyma in CT data, there is still a challenge for our lung segmentation algorithm. For use in a clinical environment, lung segmentation must be robust against pathological alterations and be fully automated. According to the section 2.1.1, we manually choose the two seed points from left and right lung region in order to implement region growing; therefore, the proposed lung segmentation algorithm is not fully automated. In future work, according to Kuhnigk [8], since the lungs usually occupy plenty of space and substantial variations in their position within the chest are unlikely, a heuristic search for air-dense areas is used. For the extraction of the pulmonary airspace regions, one seed point is usually enough, since the airway systems of both lungs are interconnected at the carina, i.e., the juncture of left and right main bronchi. However, the segmentation of both lungs from one seed point can be hindered by pathological conditions such as an obstructed left or right main bronchus, and/or a complete collapse of one of the lungs. We therefore always look for low density areas on both sides, restricting our search to the central third (in x-direction) of the data. For each lung, one seed point is determined that matches the condition of being below 600 HU [8].

4.2 Discussion of the Vessel Segmentation

The proposed algorithm can successfully segment the vasculature, while remove the isolated high-density structures such as thickened parts of the fissures and part of bronchi. The most difficult part of vessel segmentation is extracting vessel while remove the bronchi voxels. Since the method we used did not take the bronchi into account, the proposed method cannot differentiate the vasculature structure and bronchi structure, as showing in the figure 4. To obtain an accurate pulmonary vasculature, we can first segment the bronchi by the method of Lassen et al. [1] and then remove the bronchi voxel from the result of vessel segmentation.

4.3 Discussion of the Fissure Segmentation

The fissure segmentation is the hardest part of this project, since the intensity difference between fissure with the lung parenchyma is subtle and the width of fissure is only about 4-5 voxels. The proposed algorithm can successfully extract the most part of the fissure structure and get rid of the bronchi and vasculature etc. as showing in the figure 7. Comparing with the

ground truth, the proposed algorithm can obtain a relatively better result of left fissure than the two right fissures, since the right fissures are close from each other and there are more vasculature and bronchi in the right lung. Furthermore, for lobe segmentation as future work, according to Lassen et al. [1], we need to combine information from automatic segmentations of the lungs, fissures, vessels, and bronchi to segment the lobes, since this approach is anatomically inspired and similar to the way humans determine the lobar boundary. Visible fissures are used for segmentation because they are the most precise feature, but in absence of a fissure, the vessels and airways become more important. Vessels are distributed all over the lung and due to the high contrast to the lung parenchyma a good segmentation of the vessels is feasible. But in some cases vessels cross the lobar boundaries. Thus, the assumption that there are no vessels at the lobar boundary is not always correct. In contrast a deep segmentation of the bronchi is challenging but there are definitely no bronchi at the boundary between the lobes. By combining the information of different anatomical structures, we expect to get as much as possible information to perform an accurate lobe segmentation [1].

Conclusion

In this project, I implement the algorithm to segment the lung vessel and fissure for CT scans. The vessel segmentation is based on 3D connected component analysis and morphological operation, while the fissure segmentation is based on Hessian matrix analysis and vector-based region growing. The dataset I used is the LOLA11 which contains 55 cases including both normal and abnormal cases and the performance metrics is the average mean distance between the voxel in the reference and the closest voxel in the lobar segmentation. The final accuracy for the fissure segmentation is 6.23mm.

References

[1] Bianca Lassen, Eva M. van Rikxoort, Michael Schmidt, Sjoerd Kerkstra, Bram van Ginneken, and Jan-Martin Kuhnigk “Automatic Segmentation of the Pulmonary Lobes From Chest CT Scans Based on Fissures, Vessels, and Bronchi”, IEEE Transactions on Medical Imaging, Vol. 32, No. 2, Feb. 2013

[2] Li Zhang, Eric A. Hoffman, and Joseph M. Reinhardt, “Atlas-Driven Lung Lobe Segmentation in Volumetric X-Ray CT Images”, IEEE Transactions on Medical Imaging, Vol. 25, No. 1, Jan. 2006

[3] Rafael WiemkerT, Thomas Bu¨low, and Thomas Blaffert, “Unsupervised extraction of the pulmonary interlobar fissures from high resolution thoracic CT data”, International Congress Series 1281 (2005) 1121–1126

[4] E. M. van Rikxoort, B. de Hoop, S. van de Vorst, M. Prokop, and B. van Ginneken, “Automatic segmentation of pulmonary segments from volumetric chest CT scans,” IEEE Trans. Med. Imag., vol. 28, no. 4, pp. 621–630, Apr. 2009.

[5] J. Pu, B. Zheng, J. Leader, C. Fuhrman, F. Knollmann, A. Klym, and D. Gur, “Pulmonary lobe segmentation in CT examinations using im- plicit surface fitting,” IEEE Trans. Med. Imag., vol. 28, no. 12, pp. 1986–1996, Dec. 2009.

[6] E. van Rikxoort, M. Prokop, B. de Hoop, M. Viergever, J. Pluim, and B. van Ginneken, “Automatic segmentation of pulmonary lobes robust against incomplete fissures,” IEEE Trans. Med. Imag., vol. 29, no. 6, pp. 1286–1296, Jun. 2010.

[7] J.-M. Kuhnigk, V. Dicken, S. Zidowitz, L. Bornemann, B. Kuem- merlen, S. Krass, H.-O. Peitgen, S. Yuval, H.-H. Jend, W. S. Rau, and T. Achenbach, “New tools for computer assistance in thoracic CT—Part I: Functional analysis of lungs, lung lobes, and bronchopul- monary segments,” Radio Graphics, vol. 25, no. 2, pp. 525–536, 2005.

[8] Jan-Martin Kuhnigk, “Quantitative Analysis of Lung Morphology and Function in Computed Tomographic Images”, Doctoral dissertation, MeVis Research, Center for Medical Image Computing, 2008.

Appendix

A. Program Documentation

NAME

< read_data.py >-< this program is the class-LoadData- to read CT scans, including the functions to load data and showing image slice >

DISCRIPTION

LoadData(image directory, image name)

This class is to read 3D image data.

LoadData.loaddata(): Load image with given image path.

LoadData.tileimage(index1, index2): Tile the 3D image into two selected slices for showing.

LoadData.sitk_show(title=None, margin=0.0, dpi=40): Show the tiled 2D images.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

NAME

< lung_segment.py >-< This script is to segment the lung out of background as a preprocessing for lobes segmentation.>

DISCRIPTION

LungSegment(image)

This class is designed for 3D segmentation of lung.

LungSegment.conv_2_uint8(WINDOW_LEVEL=(1050,500)): Convert image to 8-bit image.

LungSegment.regiongrowing(seed_pts): Implement ConfidenceConnected by SimpleITK tools with given seed points

LoadData.image_showing (title=None): Show images.

LoadData.image_closing(size=7): Implement morphological closing to fix the "holes" inside the image.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

NAME

< vessel_segment.py >-< This script is for pulmonary vessel enhancement.>

DISCRIPTION

VesselSegment (Original image, Closing image)

This class is designed for lung vasculature enhancement.

*VesselSegment.generate_lung_mask():*Generate lung mask by setting the outside rigion as 0.

VesselSegment.downsampling(): Downsample the input image from [-1024, 0] to [0, 255] for reducing memory requirement.

VesselSegment.thresholding(thval=130): Threshold the image with given thresholding value.

VesselSegment.max_filter(filter_size=5): Implement maximum filter.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

NAME

< region_growing >-< This program is for computing 3D region growing >

SYNOPSIS

region_growing input_file_name outout_file_name threshold_value

DISCRIPTION

This program is for computing 3D region growing with given threshold value. The default value is set as 0.1. This program uses 6 neighbors for each voxel.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

NAME

< region_growing_26 >-< This program is for computing 3D region growing >

SYNOPSIS

region_growing input_file_name outout_file_name threshold_value

DISCRIPTION

This program is for computing 3D region growing with given threshold value. The default value is set as 0.1. This program uses 26 neighbors for each voxel.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

NAME

< vector_region_growing >-< This program is for computing vector-based region growing >

SYNOPSIS

vector_region_growing <input_file > <outout_file > <threshold_value>

DISCRIPTION

This program is for computing vector-based region growing with given threshold value. The default value is set as 0.9. This program uses 6 neighbors for each voxel. The program first compute the Hessian matrix for each voxel and calculate the eigenvalues for the matrix. Furthermore, it uses the eigenvector of the maximum absolute value of the corresponding eigenvalue to compute the vector-based region growing. The inner product of the eigenvectors is calculated for connecting component.

CONSTRIANTS

The type of input images can be: *.mhd, *.mha.

B. Program Code

1. read_data.py

```
"""
This file is to load the input image and convert to numpy array.
"""
import SimpleITK as sitk
import matplotlib.pyplot as plt

class LoadData:
    """
    This class is designed to load "one" input image.
    """
    def __init__(self, path, name):
        """
        :param path: image derectory
        :param name: image name
        """
        self.img_path = path + name
        self.image = None
        self.slices = None

    def loaddata(self):
        """
        Load image with given image path.
        :return: None
        """
        self.image = sitk.ReadImage(self.img_path)

    def tileimage(self, index1, index2):
        """
        Tile the 3D image into two selected slices for showing.
        :param index1: selected slice 1
        :param index2: selected slice 2
        :return: None
        """
        self.slices = sitk.Tile(self.image[:, :, index1],
                                self.image[:, :, index2],
                                (2, 1, 0))

    def sitk_show(self, title=None, margin=0.0, dpi=40):
        """
        Show the tiled 2D images.
        :param title: Title
        :param margin: Margin
        :param dpi: ???
        :return: None
        """
        nda = sitk.GetArrayFromImage(self.slices)
        figsize = (1 + margin) * nda.shape[0] / dpi, (1 + margin) * nda.shape[1] / dpi
        extent = (0, nda.shape[1], nda.shape[0], 0)
        fig = plt.figure(figsize=figsize, dpi=dpi)
        ax = fig.add_axes([margin, margin, 1 - 2 * margin, 1 - 2 * margin])

        plt.set_cmap("gray")
        ax.imshow(nda, extent=extent, interpolation=None)
        if title:
            plt.title(title)
        plt.show()
```

2. lung_segment.py

```
"""
This script is to segment the lung out of background as a prepocessing for lobes segementation.
"""
import SimpleITK as sitk
import gui

class LungSegment:
    """
```

```
    This class is designed for 3D segmentation of lung, including the methods:
    ...
    """
    def __init__(self, img):
        self.img = img
        self.temp_img = None
        self.img_uint8 = None

    def conv_2_uint8(self, WINDOW_LEVEL=(1050,500)):
        """
        Convert original image to 8-bit image
        :param WINDOW_LEVEL: Using an external viewer (ITK-SNAP or 3DSlicer)
                             we identified a visually appealing window-level setting
        :return: None
        """
        # self.img_uint8 = sitk.Cast(self.img,
        #                            sitk.sitkUInt8)
        self.img_uint8 = sitk.Cast(sitk.IntensityWindowing(self.img,
                                   windowMinimum=WINDOW_LEVEL[1] - WINDOW_LEVEL[0] / 2.0,
                                   windowMaximum=WINDOW_LEVEL[1] + WINDOW_LEVEL[0] / 2.0),
                                   sitk.sitkUInt8)

    def regiongrowing(self, seed_pts):
        """
        Implement ConfidenceConnected by SimpleITK tools with given seed points
        :param seed_pts: seed points for region growing [(z,y,x), ...]
        :return: None
        """
        self.temp_img = sitk.ConfidenceConnected(self.img, seedList=seed_pts,
                                                           numberOfIterations=0,
                                                           multiplier=2,
                                                           initialNeighborhoodRadius=1,
                                                           replaceValue=1)

    def image_showing(self, title=''):
        """
        Showing image.
        :return: None
        """
        gui.MultiImageDisplay(image_list=[sitk.LabelOverlay(self.img_uint8, self.temp_img)],
                              title_list=[title])

    def image_closing(self, size=7):
        """
        Implement morphological closing to fix the "holes" inside the image.
        :param size: the size the closing kernel
        :return: None
        """
        closing = sitk.BinaryMorphologicalClosingImageFilter()
        closing.SetForegroundValue(1)
        closing.SetKernelRadius(size)
        self.temp_img = closing.Execute(self.temp_img)
```

3. vessel_segment.py

```
"""
This file is for plumonary vessels enhancement.
"""
import numpy as np
import SimpleITK as sitk
import scipy as sp
import cv2

from itertools import izip_longest

class VesselSegment:
    """
    This class is desigened for lung vasculature enhancement, including the methods of:
                1. downsampling
                2. thresholding
                3. 3D region growing
                4. filtering small struture
    """

    def __init__(self, original, closing):
        """
        :param original: the orignal image in ITK formate
        :param closing: the Closing result image
```

```
        :param thval: threshold value (default: 130HU)
        :param filter_vol: filter value for removing small struture after region growing (default: 2ml)
        """
        self.original_img = original
        self.closing_img = closing
        self.img = None
        self.thval = None
        self.filter_vol = None
        self.temp_img = None

    def generate_lung_mask(self):
        """
        Generate lung mask
        :return: None
        """
        self.img = sitk.GetArrayFromImage(self.original_img).copy()
        self.img[self.closing_img == 0] = 0

    def downsampling(self):
        """
        Downsample the input image from [-1024, 0] to [0, 255] for reducing memory requirement.
                               / max(0, min(254, (Vorig+1024)/4), v belongs to Lung region
                      Vds =
                               \ 255,                              otherwise
        :return: None
        """
        temp = (self.img + 1024) / 4
        temp[temp > 254] = 254
        temp[temp < 0] = 0
        self.temp_img = temp

    def thresholding(self, thval=130):
        """
        Threshold the image with given thresholding value.
        :return: None
        """
        self.thval = thval
        self.temp_img[self.temp_img < thval] = thval

    def max_filter(self, filter_size=5):
        """
        Implement maximum filter.
        :param filter_size: filter size
        :return: None
        """
        temp = self.temp_img.copy()
        temp[temp >= 254] = 0
        temp[temp <= self.thval] = 0
        self.temp_img = sp.ndimage.filters.maximum_filter(temp, size=filter_size)
```

4. region_growing.cxx

```
#include <iostream>
#include <string>
#include "stdlib.h"

#include "itkImageRegionIterator.h"
#include "itkImageRegionConstIterator.h"
#include "itkImageRegionIteratorWithIndex.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImportImageFilter.h"

const unsigned short dimension = 3;

typedef float floatVoxelType;
typedef itk::Image<floatVoxelType, dimension> ImageType;
typedef itk::Point< float, ImageType::ImageDimension > PointType;
typedef itk::ImageRegionIterator<ImageType> OutputIterType;

class Regiongrowing {
public:
    ImageType::Pointer OutputImage;

    unsigned int xs;
    unsigned int ys;
    unsigned int zs;
```

```
    Regiongrowing(const ImageType::Pointer& image, float thresh_val) {
        // This method is to compute the 3D region growing
        // :params image: the 3D lung mask image
        // :params thresh_val: the threshold value for region growing
        OutputImage = ImageType::New();
        OutputImage->SetRegions( image->GetLargestPossibleRegion() );
        OutputImage->Allocate(true); // initialize buffer to zero

        OutputIterType OutputIter(OutputImage,image->GetLargestPossibleRegion());

        int shape[3];
        xs = image->GetLargestPossibleRegion().GetSize()[0];
        ys = image->GetLargestPossibleRegion().GetSize()[1];
        zs = image->GetLargestPossibleRegion().GetSize()[2];

        // compute region growing
        float label = 1;
        for (int z = 0; z < zs; z++){
          for (int y = 0; y < ys; y++){
            for (int x = 0; x < xs; x++){
              PointType point;
              point[0] = x;
              point[1] = y;
              point[2] = z;
              ImageType::IndexType out_index;

              image->TransformPhysicalPointToIndex( point, out_index );
              ImageType::PixelType p_vec = image->GetPixel(out_index);
              if (OutputImage->GetPixel(out_index) == 0 && p_vec != 0){
                int volume = 1;
                int *vol_point = &volume;
                this->region_growing(image, point, label, thresh_val, vol_point);
                label = label + 1;
              }
            }
          }
        }
    }
    void region_growing(const ImageType::Pointer& , PointType, float, float, int *);
};
void Regiongrowing::region_growing(const ImageType::Pointer& image, PointType point, float label, float thresh_val,
int *vol_point){
  //This function is to compute region growing.
  // :params image: the image for region growing
  // :params point: the point for region growing
  // :params label: point label
  // :params thresh_val: the threshold value
  // :params vol_point: the volume for given label to limit the number of recursive.
  float x = point[0]; float y = point[1]; float z = point[2];

  PointType neighbors[6];
  neighbors[0][0] = x+1; neighbors[1][0] = x-1; neighbors[2][0] = x; neighbors[3][0] = x; neighbors[4][0] = x;
neighbors[5][0] = x;
  neighbors[0][1] = y; neighbors[1][1] = y; neighbors[2][1] = y+1; neighbors[3][1] = y-1; neighbors[4][1] = y;
neighbors[5][1] = y;
  neighbors[0][2] = z; neighbors[1][2] = z; neighbors[2][2] = z; neighbors[3][2] = z; neighbors[4][2] = z+1;
neighbors[5][2] = z-1;

  ImageType::IndexType p_index;
  image->TransformPhysicalPointToIndex( point, p_index );
    ImageType::PixelType p = image->GetPixel( p_index );

  for (int i = 0; i < 6; i++){
    if (neighbors[i][0] >= 0 && neighbors[i][0] < xs &&
        neighbors[i][1] >= 0 && neighbors[i][1] < ys &&
        neighbors[i][2] >= 0 && neighbors[i][2] < zs){
      ImageType::IndexType out_index;
      image->TransformPhysicalPointToIndex( neighbors[i], out_index );
      if (OutputImage->GetPixel(out_index) == 0){
        ImageType::PixelType np = image->GetPixel( out_index );
        float diff = std::abs(np - p);
        if (np > 0 && *vol_point < 20000){
          OutputImage->SetPixel( out_index, label);
          *vol_point = *vol_point + 1;
          this->region_growing(image, neighbors[i], label, thresh_val, vol_point);

        }
      }
```

```
    }
  }
}

int main( int argc, char * argv[] )
{
  time_t tStart = clock();

  if( argc < 2 ) {
      std::cerr << "Usage: " << std::endl;
      std::cerr << argv[0] << "  inputImageFile  outputImageFile" << std::endl;
      return EXIT_FAILURE;
  }

  typedef itk::ImageFileReader< ImageType >  readerType;

  float thresh = 0.1;

  // Read Image
  readerType::Pointer reader = readerType::New();
  reader->SetFileName( argv[1] );
  reader->Update();

  // Compute eigenvalues
  std::cout << "  Region Growing " << std::endl;
  Regiongrowing rg = Regiongrowing::Regiongrowing( reader->GetOutput(), thresh );

  std::cout << "   Saving Image..." << std::endl;
  typedef itk::ImageFileWriter < ImageType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName( argv[2] );
  writer->SetInput( rg.OutputImage );

  writer->Update();
  printf("Time taken: %.2fs\n", (float)(clock() - tStart)/CLOCKS_PER_SEC);

  return EXIT_SUCCESS;
}
```

5. vector_region_growing.cxx

```
#include <iostream>
#include <string>
#include "stdlib.h"
#include <math.h>
#include <unordered_set>

#include "itkImageRegionIterator.h"
#include "itkImageRegionConstIterator.h"
#include "itkImageRegionIteratorWithIndex.h"
#include "itkHessianRecursiveGaussianImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImportImageFilter.h"
#include "itkMedianImageFilter.h"

const unsigned short dimension = 3;
typedef int VoxelType;
typedef float outVoxelType;
typedef itk::Image<VoxelType, dimension> InputImageType;
typedef itk::Image<outVoxelType, dimension> OutputImageType;
typedef itk::Point< float, InputImageType::ImageDimension > PointType;

typedef float HessianVoxelType;
typedef itk::Image<HessianVoxelType, dimension> HessianInnerImageType;
typedef itk::CastImageFilter<InputImageType, HessianInnerImageType> CastFilterType;

typedef itk::HessianRecursiveGaussianImageFilter<HessianInnerImageType> HFilterType;
typedef itk::Vector<HessianVoxelType, dimension> VecVoxType;
typedef itk::Matrix<HessianVoxelType, dimension, dimension> MatVoxType;
typedef itk::Image<VecVoxType, dimension> VecEigHImageType;
typedef itk::Image<MatVoxType, dimension> MatEigHImageType;

typedef itk::MedianImageFilter<OutputImageType, OutputImageType > MedianImageFilterType;
typedef itk::ImportImageFilter< outVoxelType, dimension >   ImportFilterType;
typedef itk::ImageRegionIterator<OutputImageType> OutputImageIterType;
typedef itk::ImageRegionIterator<VecEigHImageType> OutputIterType;
```

```
typedef itk::ImageRegionIterator<InputImageType> InputImageIterType;
typedef itk::SymmetricEigenAnalysis<MatVoxType, VecVoxType, MatVoxType> EigValAnalysisType;
typedef MatEigHImageType::Pointer MatEigHImagePointerType;
typedef MatEigHImageType::RegionType MatRegionType;
typedef MatEigHImageType::PointType MatPointType;
typedef MatEigHImageType::SpacingType MatSpacingType;

typedef VecEigHImageType::Pointer VecEigHImagePointerType;
typedef itk::ImageRegionIteratorWithIndex<HFilterType::OutputImageType> HIterType;

class cmp
{
public:
    bool operator() (  HessianVoxelType a,  HessianVoxelType b )
    {
        return std::abs(a) < std::abs(b) ;
    }
};

class EigValHessian {
public:
    MatRegionType region;
    MatSpacingType spacing;
    MatPointType origin;

    CastFilterType::Pointer CastFilter;
    HFilterType::Pointer HFilter;

    EigValAnalysisType EigValAnalysis;

    VecEigHImagePointerType VecEigHImagePointer;

    OutputImageType::Pointer OutputImage;

    ImportFilterType::Pointer importFilter;

    std::unordered_set<outVoxelType> label_set;

    MedianImageFilterType::Pointer medianFilter;

    EigValHessian(const InputImageType::Pointer& image, float sigma, float alpha, float beta, float gama) {
        // This method is to compute the Hessian matrix by ITK filter: HessianRecursiveGaussianImageFilter
        // Furthermore, calculte the fissure structure measurement with given parameters and then using vector
        // based region growing method to segment fissures.
        // :params image: the 3D lung mask image
        // :params sigma: the standard deviation for Gaussian function in Hessian matrix
        // :params alpha, beta, gama: parameters for fissure representation
        VecVoxType EigVal;
        MatVoxType EigMat,tmpMat;
        for(int i=0;i<3;i++)
            for(int j=0;j<3;j++)
                EigMat[i][j]=0;

        region=image->GetLargestPossibleRegion();
        spacing=image->GetSpacing();
        origin=image->GetOrigin();

        // initialize the Hassian filter
        CastFilter = CastFilterType::New();
        HFilter = HFilterType::New();
        HFilter->SetSigma(sigma);

        EigValAnalysis.SetDimension(3);
        CastFilter->SetInput(image);
        HFilter->SetInput(CastFilter->GetOutput());

        printf("Processing HFilter\n");
        HFilter->Update();

        VecEigHImagePointer=VecEigHImageType::New();
        VecEigHImagePointer->SetRegions(region);
        VecEigHImagePointer->SetOrigin(origin);
        VecEigHImagePointer->SetSpacing(spacing);
        VecEigHImagePointer->Allocate();

        EigVal[0]=EigVal[1]=EigVal[2]=0;
        VecEigHImagePointer->FillBuffer(EigMat[0]);

        OutputImage = OutputImageType::New();
```

```
OutputImage->SetRegions( region );
OutputImage->Allocate(true); // initialize buffer to zero

HIterType HIter(HFilter->GetOutput(),region);

OutputIterType OutputIter(VecEigHImagePointer,region);

itk::SymmetricSecondRankTensor<float,3> Tensor;

bool fissure_cond = true;

InputImageIterType InputImageIter(image, region);

// this is the mean and std value for vessel which is compute according to
// histogram analysis in previous result.
outVoxelType mean = 198.275350;
outVoxelType std = 42.571917;

outVoxelType vessel_thesh = mean - 3 * std;

for(HIter.GoToBegin(),OutputIter.GoToBegin(),InputImageIter.GoToBegin();
    !HIter.IsAtEnd()&&!OutputIter.IsAtEnd()&&!InputImageIter.IsAtEnd();
    ++HIter,++OutputIter,++InputImageIter){
    Tensor=HIter.Get();
    tmpMat[0][0]=Tensor[0];
    tmpMat[0][1]=Tensor[1];
    tmpMat[1][0]=Tensor[1];
    tmpMat[0][2]=Tensor[2];
    tmpMat[2][0]=Tensor[2];
    tmpMat[1][1]=Tensor[3];
    tmpMat[2][1]=Tensor[4];
    tmpMat[1][2]=Tensor[4];
    tmpMat[2][2]=Tensor[5];

    // compute the eigenvalues given a 3*3 Hessian matrix.
    EigValAnalysis.ComputeEigenValuesAndVectors(tmpMat,EigVal,EigMat);

    // obtain the maximum absolute value for eigenvalues
    HessianVoxelType sortedEigVal[3] = {EigVal[0],EigVal[1],EigVal[2]};

    std::sort(sortedEigVal, sortedEigVal+3, cmp());

    // Compute Structure
    HessianVoxelType theta;
    if (sortedEigVal[2] >= 0){
      theta = 0;
    }else{
      theta = 1;
    }
    HessianVoxelType Fstructure, Fsheet, Sfissure;
    Fstructure = theta * exp(-1*(pow((std::abs(sortedEigVal[2])-alpha)/beta,6)));
    Fsheet = exp(-1*(pow(sortedEigVal[1]/gama,6)));

    Sfissure = Fstructure * Fsheet;

    // Convert to uint8 value
    VoxelType pixel_val = (InputImageIter.Get() + 1024) / 4;

    // Thresholding the result fissure structure measurement
    fissure_cond = Sfissure > 0.8 && pixel_val < vessel_thesh ? true : false;

    // Save the corresponding eigenvector for the maximum eigenvalue
    for (int i = 0; i < 3; i++){
      if (EigVal[i] == sortedEigVal[2] && fissure_cond){
        OutputIter.Set(EigMat[i]);
        break;
      }
    }

}

printf("Processing computing eigenvalues and eigenvectors\n");
VecEigHImagePointer->Update();
printf("Finish computing eigenvalues and eigenvectors\n");

// Compute the vector based region growing with given eigenvector.
const OutputImageType::RegionType region = VecEigHImagePointer->GetBufferedRegion();
const OutputImageType::SizeType size = region.GetSize();
const unsigned int xs = size[0];
```

```
        const unsigned int ys = size[1];
        const unsigned int zs = size[2];
        float label = 1;
        for (int z = 0; z < zs; z++){
          for (int y = 0; y < ys; y++){
            for (int x = 0; x < xs; x++){
              PointType point;
              point[0] = x;
              point[1] = y;
              point[2] = z;
              OutputImageType::IndexType out_index;
              VecEigHImageType::IndexType vec_index;

              OutputImage->TransformPhysicalPointToIndex( point, out_index );
              VecEigHImagePointer->TransformPhysicalPointToIndex( point, vec_index );
              VecEigHImageType::PixelType p_vec = VecEigHImagePointer->GetPixel(vec_index);

              if (OutputImage->GetPixel(out_index) == 0 && p_vec[0]+p_vec[1]+p_vec[2] != 0){
                int volume = 1;
                int *vol_point = &volume;
                this->region_growing(point, label, vol_point);
                label = label + 1;
              }
            }
          }
        }
    }
    void region_growing(PointType, float, int *);
};
void EigValHessian::region_growing(PointType point, float label, int *vol_point){
  // This function is to compute the vector based region growing.
  // :params point: the point for region growing
  // :params label: point label
  // :params vol_point: the volume for given label to limit the number of recursive.
  int x = point[0]; int y = point[1]; int z = point[2];
  const OutputImageType::RegionType region = VecEigHImagePointer->GetBufferedRegion();
  const OutputImageType::SizeType size = region.GetSize();
  const unsigned int xs = size[0];
  const unsigned int ys = size[1];
  const unsigned int zs = size[2];

  PointType neighbors[6];
  neighbors[0][0] = x+1; neighbors[1][0] = x-1; neighbors[2][0] = x; neighbors[3][0] = x; neighbors[4][0] = x;
neighbors[5][0] = x;
  neighbors[0][1] = y; neighbors[1][1] = y; neighbors[2][1] = y+1; neighbors[3][1] = y-1; neighbors[4][1] = y;
neighbors[5][1] = y;
  neighbors[0][2] = z; neighbors[1][2] = z; neighbors[2][2] = z; neighbors[3][2] = z; neighbors[4][2] = z+1;
neighbors[5][2] = z-1;

  VecEigHImageType::IndexType p_index;
  VecEigHImagePointer->TransformPhysicalPointToIndex( point, p_index );
    VecEigHImageType::PixelType p = VecEigHImagePointer->GetPixel( p_index );

  for (int i = 0; i < 6; i++){
    if (neighbors[i][0] >= 0 && neighbors[i][0] < xs &&
        neighbors[i][1] >= 0 && neighbors[i][1] < ys &&
        neighbors[i][2] >= 0 && neighbors[i][2] < zs){
      OutputImageType::IndexType out_index;
      OutputImage->TransformPhysicalPointToIndex( neighbors[i], out_index );
      if (OutputImage->GetPixel(out_index) == 0){
        VecEigHImageType::IndexType vec_index;
        VecEigHImagePointer->TransformPhysicalPointToIndex( neighbors[i], vec_index );
        VecEigHImageType::PixelType np = VecEigHImagePointer->GetPixel( vec_index );
        float product = np[0] * p[0] + np[1] * p[1] + np[2] * p[2];
        if (product > 0.9){
          *vol_point = *vol_point + 1;
          OutputImage->SetPixel( out_index, label);
          this->region_growing(neighbors[i], label, vol_point);
        }
      }
    }
  }
}

int main( int argc, char * argv[] )
{
  time_t tStart = clock();

  if( argc < 2 ) {
```

```
        std::cerr << "Usage: " << std::endl;
        std::cerr << argv[0] << "  inputImageFile  outputImageFile" << std::endl;
        return EXIT_FAILURE;
    }

    typedef itk::ImageFileReader< InputImageType >  readerType;

    float sigma = 0.5;
    float alpha = 90;
    float beta = 120;
    float gama = 100;

    // Read Image
    readerType::Pointer reader = readerType::New();
    reader->SetFileName( argv[1] );
    reader->Update();

    // Compute eigenvalues
    std::cout << "  Compute Eigenvalues " << std::endl;
    EigValHessian eigenvalues = EigValHessian::EigValHessian( reader->GetOutput(), sigma, alpha, beta, gama );

    std::cout << "   Saving Image..." << std::endl;
    typedef itk::ImageFileWriter < OutputImageType > WriterType;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName( argv[2] );
    writer->SetInput( eigenvalues.OutputImage );

    writer->Update();
    printf("Time taken: %.2fs\n", (float)(clock() - tStart)/CLOCKS_PER_SEC);

    return EXIT_SUCCESS;
}
```