January 2016

Vascular Tree Structure: Fast Curvature Regularization and Validation

Egor Chesakov
The University of Western Ontario

Supervisor
Dr. Yuri Boykov
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

VASCULAR TREE STRUCTURE: FAST CURVATURE REGULARIZATION AND VALIDATION

(Thesis format: Monograph)

by

Egor Chesakov

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

This work addresses the challenging problem of accurate vessel structure analysis in high resolution 3D biomedical images. Typical segmentation methods fail on recent micro-CT data sets resolving near-capillary vessels due to limitations of standard first-order regularization models. While regularization is needed to address noise and partial volume issues in the data, we argue that extraction of thin tubular structures requires higher-order curvature-based regularization. There are no standard segmentation methods regularizing surface curvature in 3D that could be applied to large 3D volumes. However, we observe that standard measures for vessels structure are more concerned with topology, bifurcation angles, and other parameters that can be directly addressed without segmentation. We propose a novel methodology reconstructing tree structure of the vessels using a new centerline curvature regularization technique. Our high-order regularization model is based on a recent curvature estimation method. We developed a Levenberg-Marquardt optimization scheme and an efficient GPU-based implementation of our algorithm. We also propose a validation mechanism based on synthetic vessel images. Our preliminary results on real ultra-resolution micro CT volumes are promising.

Keywords: medical imaging, vascular tree, centerline estimation, curvature regularization, Fréchet distance, validation

Acknowledgments

First of all, I would like to express my deepest gratitude to my advisor, Professor Yuri Boykov for his support and enthusiasm. He always can explain complex ideas in simple way. Thanks to him I understood now how graph-cuts work and what is the difference between submodular and supermodular set functions. I look forward to learn even more from him during PhD studying.

I appreciate the hard work of the thesis committee, professors Maria Drangova, Marc Moreno Maza and John Barron. Their advises helped me a lot to improve this work before the final submission. Specifically, I would like to thank Professor Marc Moreno Maza for explaining the GPU acceleration techniques and the concurrent algorithm analysis, and Professor John Barron for spending enormous amount of time for correcting mistakes during the revision of this thesis. I also would like to give special thanks to Professor Maria Drangova for a lot of good advice and for providing us with ultra high resolution micro-CT data. Indeed, this project would not even exist without this unique data.

I would like to thank other members of Computer Vision Group, including Dr. Olga Veksler, Dr. Lena Gorelick, Dmitri Marin, Yuchen Zhong, Hossam Isaac and Meng Tang for their invaluable help. Special thanks to Yuchen, who helped me a lot during my first days in Canada and his contribution to this work.

Last but not least, I would like to thank my family – my parents, sister and my lovely wife Katy. I would not be able to come to Canada without their huge support, understanding and love they are giving to me.

Contents

List of Figures

List of Appendices

Chapter 1

Introduction

3D imaging is playing an increasingly important role in modern diagnostics and medical research. Modern imaging methods, such as computed tomography and magnetic resonance are capable to produce 3D images at microscopic resolution. Despite availability of various medical imaging software, exploration of certain parts of the body, such as blood vessels, remains a challenging task. For example, Figure 1.1 shows a recent 3D CT cardiac image that can resolve vessels at a capillary level. As a consequence of super high resolution in new acquisition methods, there is a need for highly accurate robust analysis methods.

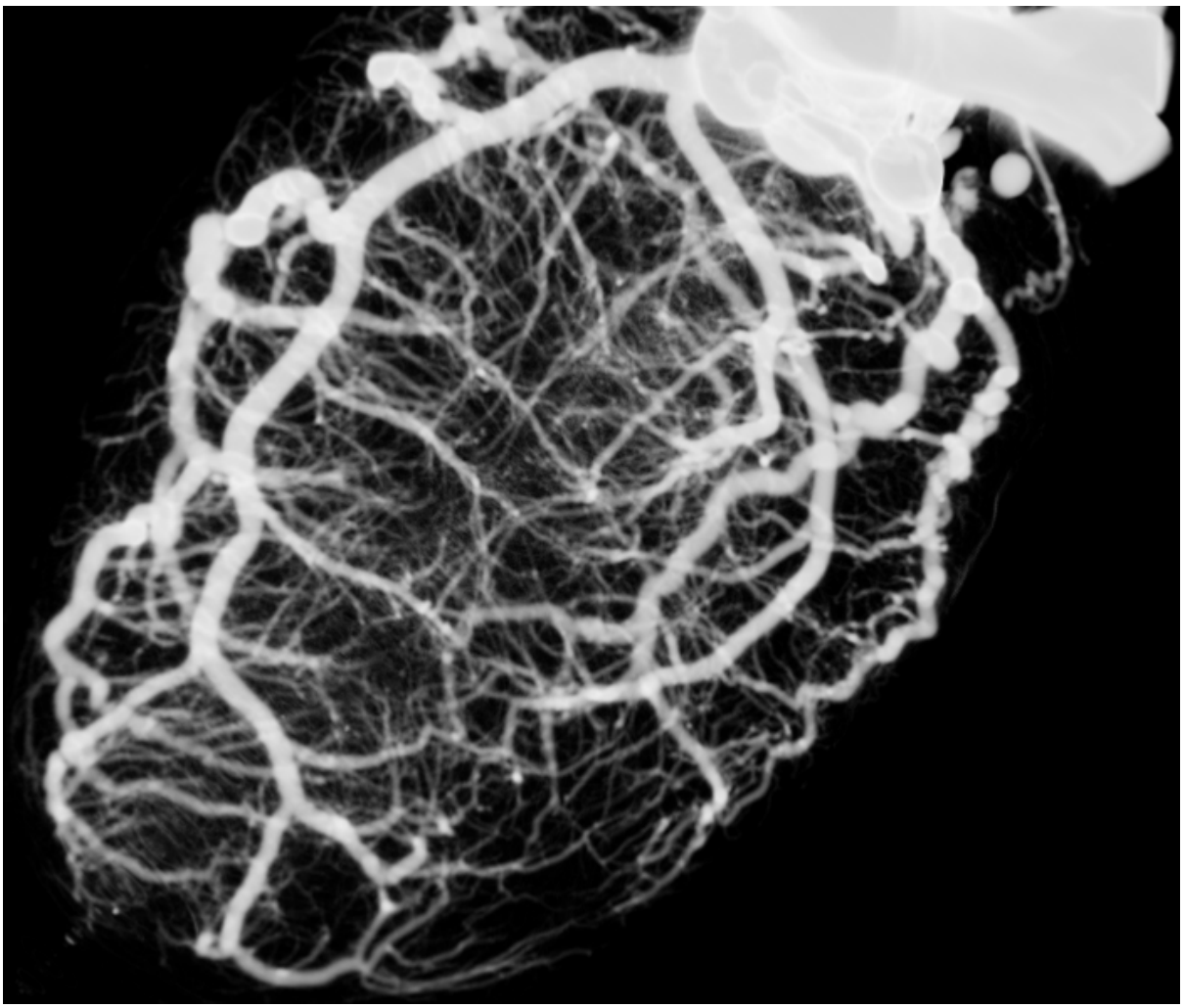
This chapter outlines the biomedical motivation for reconstruction of vascular tree structure from the state-of-the-art 3D volumes (Section 1.1) and describes computational challenges for the image analysis algorithms (Section 1.2). Our technical contributions are described in Section 1.3.

1.1 Biomedical Vascular Imaging

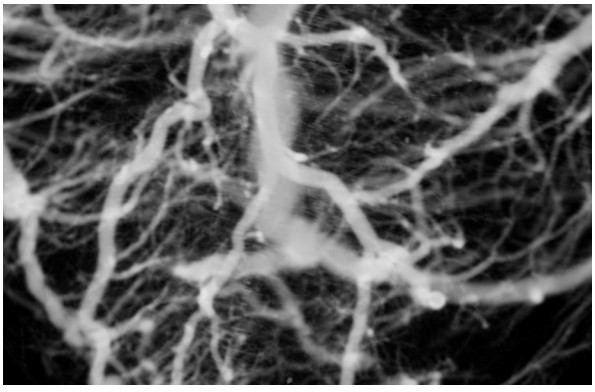
This section outlines the use of modern biomedical imaging modalities (Subsection 1.1.1) for vascular analysis applications (Subsection 1.1.2). We also discuss main technical challenges arising in the specific context of recent ultra high resolution micro-CT data (Subsection 1.1.3).

1.1.1 Imaging Modalities

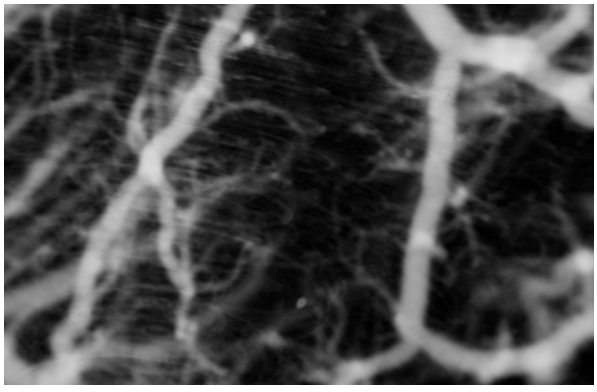
There are different imaging modalities used for blood vessels analysis. X-ray is the oldest medical imaging method. Two-dimensional images with x-ray are taken by sending a small dose of ionizing radiation though the body. X-ray has been used for many years to explore large blood vessels.



(a)



(b)



(c)

Figure 1.1: Visualization of mouse cardiac micro-CT scan. The bright voxels correspond to blood vessels and the dark voxels correspond to other tissue (e.g. heart muscles). (a) The whole volume. (b,c) Zoomed-in views of the volume demonstrate capability of micro-CT to resolve capillary vessels.

Magnetic Resonance Imaging (MRI) uses a very strong magnetic field and radio waves to produce three-dimensional images of the inside of the body. MRI can produce additional information about the structures in the body that can be obtained using a standard x-ray, or computed tomography. Diffusion MRI, also referred to as diffusion tensor imaging (DTI) produces three-dimensional *tensor images* (i.e. images whose voxels are represented by *tensors*, multidimensional arrays of numerical values).

While the vascular analysis methods we propose in this work are general, we primarly apply them to Computed Tomography (CT). This imaging modality generates 3D volumes of visual data from 2D x-ray images taken from different positions. Modern CT imaging can produce 3D views showing excellent detail of certain parts of the body, such as blood vessels and the heart. Current microcomputed tomography (micro-CT, μCT) is able to provide ultra high resolution (voxel size $\leq 20\mu m^3$).

1.1.2 Applications

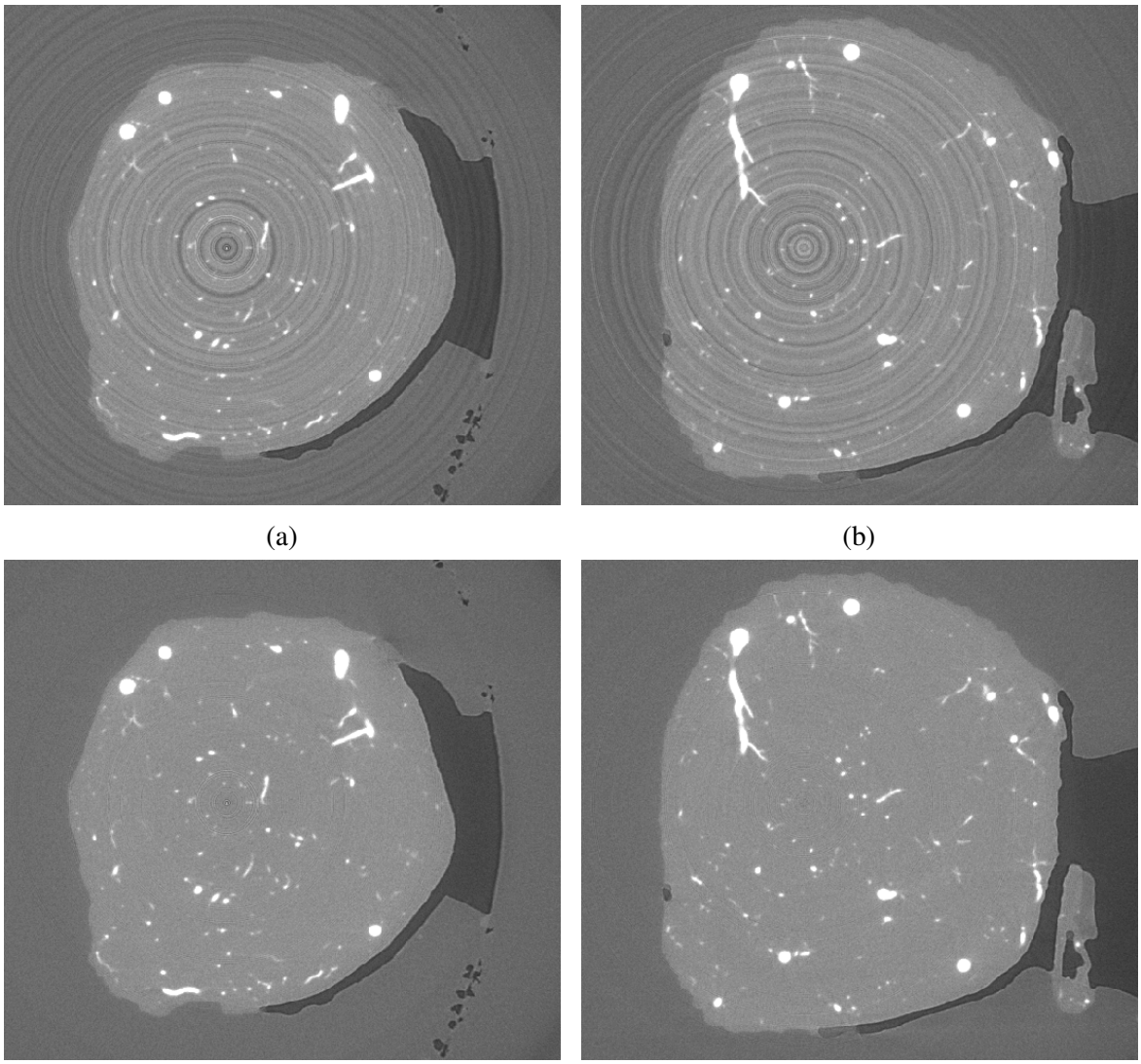
Advanced analysis of vasculature requires accurate measurement of its properties, such as the vessel diameter and length, the angle between bifurcating branches, and the topological structure of the tree [13, 14]. Existing methods, such as optical measurement are extremely time-consuming processes and demanding a large amount of manual work [13]. Consequently, there is a need for a more efficient vascular tree extraction framework. The method has to be robust with respect to the wide range of vessels scales and the complicated topological structure of vasculature.

1.1.3 Technical Problems

CT image generation is nonideal process and almost every image acquired by CT scanner contains various artifacts. There are various causes of image artifacts. One of the most common is an incorrect calibration [1]. There are methods that can either reduce artifacts, or prevent them. Artifact avoidance are mainly the concern of CT manufacturers. These methods are proprietary and are not publicly available. On the other hand, CT artifact reduction is an active research area.

Common ring artifacts are induced by CT hardware. As suggested by their name, they appear as rings centered around some axis in the middle of the volume. Figure 1.2a and Figure 1.2b demonstrate ring artifacts on cross-sectional slices of computed-tomography volume.

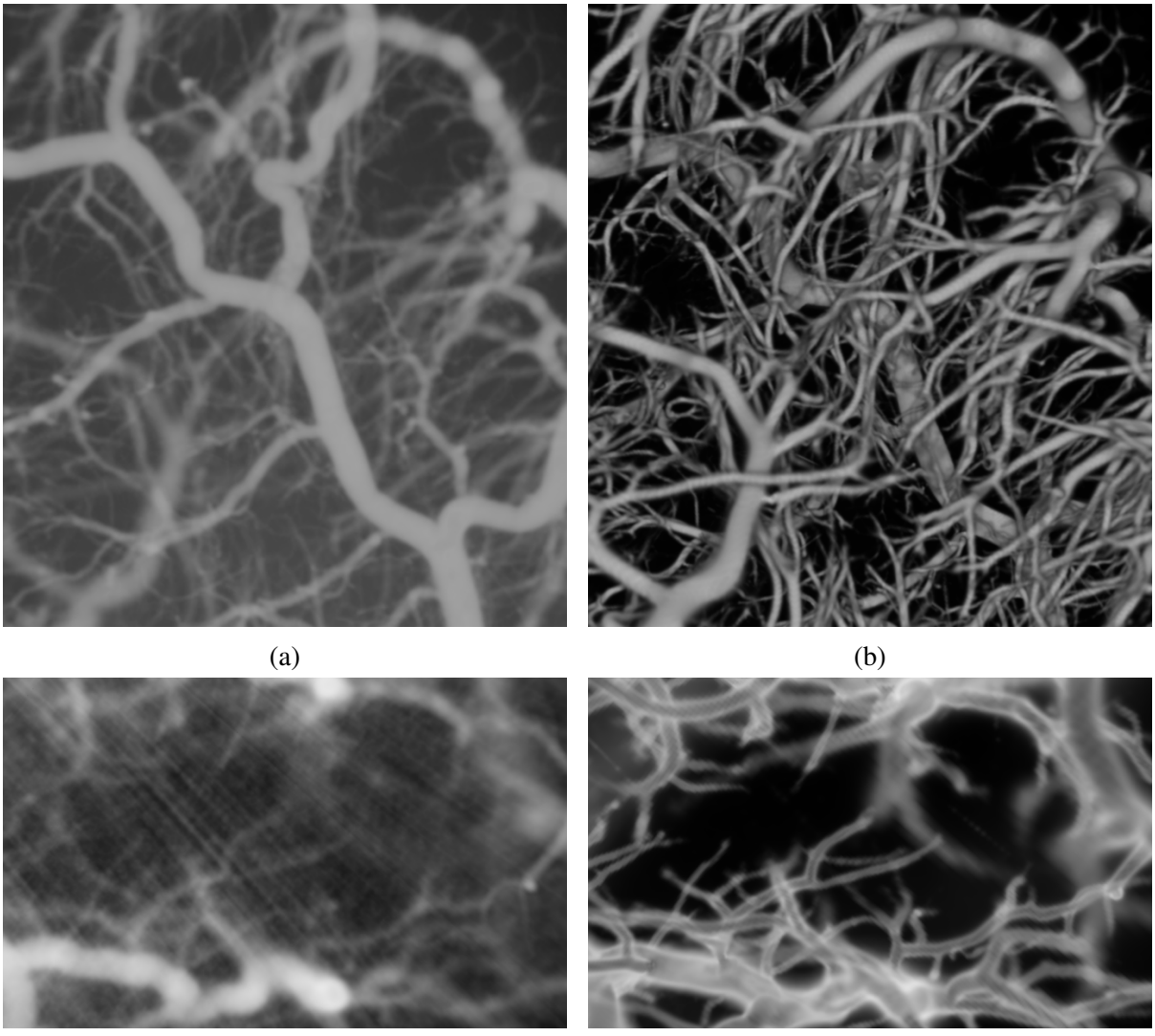
In general, the newer microscopy CT imaging generates data with significantly improved signal-to-noise ratios (SNR). However, such data is needed to solve significantly more challenging image analysis problems. In particular, the goal is to analyze blood vessels at near



(a) (b)

(c) (d)

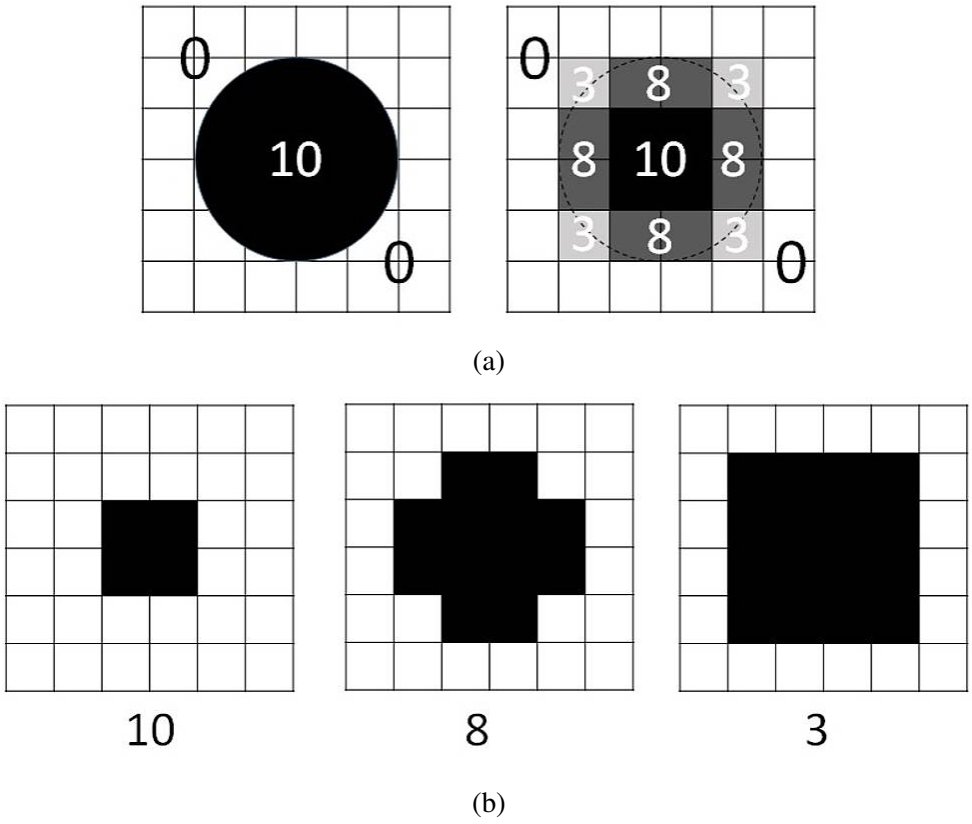
Figure 1.2: Ring acquisition artifacts caused by improperly calibrated CT scanner [1]. (a,b) Concentric circles on cross-sectional slices of micro-CT scan, also known as ring artifacts. (c,d) Result of ring reduction algorithm proposed by Sijbers and Postnov [2].



(a) (b)

(c) (d)

Figure 1.3: (a,c) Small contrast between thin vessels and the background makes it practically impossible to distinguish them by raw intensity value alone. (b,d) Vessel enhancement filtering [3] significantly increases contrast between vessels and the background, but this is not good enough for near-capillary vessels with severe partial voluming as it is shown in (d) and in Figure 1.5.



(a)

(b)

Figure 1.4: The partial volume effect is caused due to the limited resolution of the acquiring system. (a) The pixels on the border between the black circle and the white background at given pixel resolution appear at certain gray value. (b) Using of simple thresholding can not segment such circle for any threshold level. [1]

capillary level. While this was practically impossible before, even the new micro CT data have relatively weak signals for such vessels corresponding to low SNR with respect to background. As seen in Figures 1.3a and 1.3c, it is hard to distinguish thin near-capillary vessels by intensity value alone.

This issue is exacerbated by partial volume effect due to the limited spatial resolution of the scanning system (see Figure 1.4). Since each voxel of a CT image represents an average signal value within voxel's volume, such averaging could mix intensities of different tissues near organ boundaries. For example, the partial volume effect manifests itself as a blur on the boundary of blood vessel. This problem is particularly severe for thin vessels making it practically impossible to accurately segment the surface of near-capillary vessels.

[0] http://www.scanco.ch/index.php?id=299

1.2 Vascular Analysis Overview

This section reviews standard general computational methods used in vascular analysis. The noise reduction methods are described in Subsection 1.2.1. Standard segmentation methods and their limitations are discussed in Subsection 1.2.2. This section also motivates our *centerline extraction* approach that avoids explicit vessel surface segmentation.

1.2.1 Noise Reduction

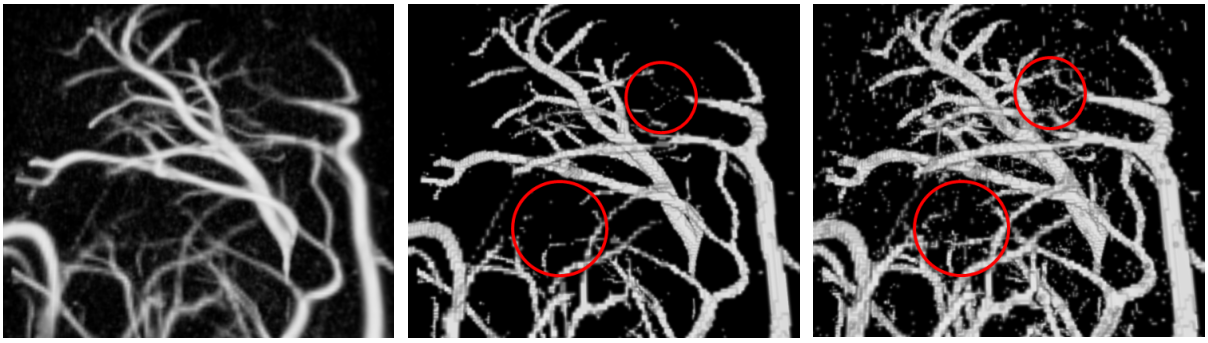
Special filtering procedures are commonly used to pre-process CT volumes in order to improve SNR and reduce various specific artifacts discussed earlier. Such pre-filtering techniques can significantly improve the quality of vessel structure extraction algorithms.

Ring artifacts can be reduced while preserving the boundaries of thin vessels by the algorithm proposed by Sijbers and Postnov [2]. Main idea of this algorithm is to transform an image to polar coordinates, where ring artifacts become line artifacts. Then, a mean or a median filter is applied to reduce them. Next, the difference of the images before and after such filtering is used to do the final correction. Finally, image is transformed back to Cartesian coordinates. Figure 1.2c and Figure 1.2d show two results of applying this algorithm.

Multiscale vessel enhancement filter by Frangi *et al.* [3] is commonly used to increase the signal-to-noise ratio. This standard method significantly improves vessel data contrast by emphasizing tubular structures and de-emphasizing everything else. The details of this method are discussed in Section 2.1. While this method is very effective, see Figure 1.3b, it is not sufficient to completely disambiguate the vessels from the background and additional robust regularization techniques are necessary to extract the vessel structure. In particular, this filter does not work well for near-capillary vessels with severe partial volume problems (as seen in Figure 1.3d). Moreover, such *vesselness* filters may create new artifacts, *e.g.* small *gaps* near bifurcations. Indeed, vesselness filter [3] is designed to respond specifically to tubular structures, but bifurcations deviate from this model.

1.2.2 Centerline Extraction vs Segmentation

One straightforward approach to vessel extraction would be volumetric segmentation. For example, it is possible to apply a number of standard segmentation techniques for computing a binary mask separating interior and exterior of the vessels. The simplest methods (e.g. thresholding or region growing) fail due to partial volume at thinner vessels (see Figure 1.5). It is technically possible to apply standard first-order regularization methods robust to weak-contrast boundaries (e.g. graph cuts, level-sets). But, such methods optimize the weighted



(a) (b) (c)

Figure 1.5: The simplest segmentation method (binary thresholding) fails due to partial volume effect at thin vessels. (a) The original volume. (b) Higher threshold removes thin vessels (e.g. those inside the red circles) while (c) lower threshold gets the vessels back, but keeps the noise.

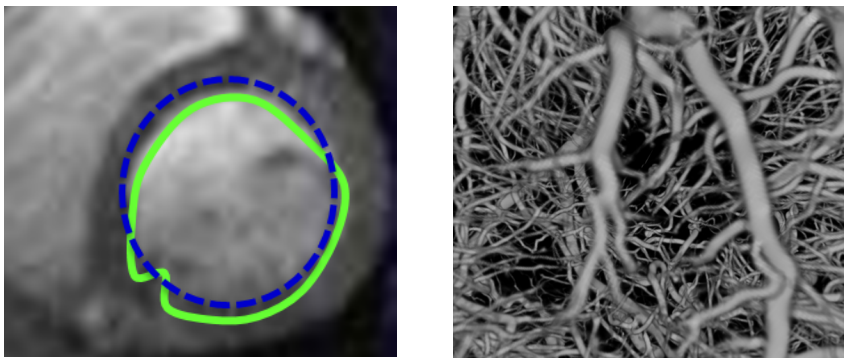
surface area of the segmentation boundary and have implicit bias to compact shapes, blobs, or spheres[15], which is also known as *shrinking bias* (see Figure 1.6). Therefore, the use of common first-order regularization methods for segmenting thin elongated structures (see Figure 1.6b) is fundamentally flawed.

We believe that second-order surface regularization minimizing (Gaussian) curvature would be appropriate for vessel segmentation. While it is possible to use second-order deformable models, this would require good initialization (e.g. see Figure 1.6a) based on known vessel topology, which is unrealistic for large complex vessel trees (see Figure 1.6b). In fact, estimation of the tree structure is our main problem. While there are curvature regularization segmentation methods [16, 17, 18, 19], they are too expensive for large 3D volumes and they have strong discretization artifacts due to restricted angular resolutions on grids or grid complexes (see Figures 1.7a and 1.7b).

We propose a method directly estimating the vessel tree by regularizing the curvature of vessel centerline using the energy formulation introduced by Marin *et al.* [20], rather than the curvature of the vessel surface (see Figure 1.7c). While our approach is related to medial axis estimation methods [21], they require segmentation of the object boundary. Our approach avoids solving the segmentation problem explicitly.

1.2.3 Related Work

The majority of vessel analysis methods work with the volumetric output of the various vesselness filters, commonly run in scale spaces. Some vessel extraction methods apply thresholding [22, 23] or region growing [24, 25]. As outlines in Figure 1.5, such methods are limited to relatively thick vessels where signal is much stronger than the noise. There is a large body of



(a) compact shape (ventricle) (b) non-compact object (vessels)

Figure 1.6: First-order segmentation methods using non-parametric surface representation (e.g. graph-cuts [4] or level-sets [5]) regularize weighted length (or surface area in 3D) of segmentation boundary. These methods have implicit bias to blobs or spheres appropriate for compact shapes like the heart ventricle in (a). Such first-order non-parametric methods display well-known *shrinking bias* for thin structures like in (b). Note that standard parametric deformable surface methods [6] require good initialization like the dashed blue circle in (a), which is practically impossible in case of complex tree structure with unknown topology (b).

(a) non-compact shape (drawing) (b) min-curvature segmentation (c) min-curvature center-lines

Figure 1.7: Higher-order non-parametric methods using curvature-regularization of the boundary would be appropriate for segmenting thin structures like vessels in Figure 1.6b or an object of interest in (a). However, the state-of-the-art second-order segmentation methods, e.g. [7], are very slow even for 2D images. Moreover, they produce strong discretization artifacts (b) due to restricted angular resolutions. In contrast, our method directly estimates object center-line by regularizing its curvature (c).

regularization methods attempting to segment relatively weak vessels using first-order regularization of their boundary [26, 27, 28, 29]. Such methods can use only very weak regularization, since first-order smoothness corresponds to surface minimization producing shrinking bias inappropriate for thin structures, see Figure 1.6. A useful overview of common regularization methods for vessels can be found in [29].

Second-order surface regularization [7] is largely out-of-reach for large volumetric data due to efficiency and artifacts, see Figure 1.7. Moreover, we are not aware of methods for Gaussian curvature required for tubular structures in 3D.

Another way to cope with limitations of standard regularization methods is to use interactive methods [30]. Such methods, as well as the standard deformable model methods, are not feasible for our high-resolution data since good initialization is not possible for detailed analysis of near-capillary vessels.

We use curvature to directly regularize the centerline after Canny-like extraction. One interesting related high-order method [23] uses curvature-like regularization for binary partitioning of arteries from veins after initial thresholding. In contrast, we use curvature to regularize the centerline.

1.3 Contributions

Our vascular tree extraction framework (see Figure 1.8) uses several standard algorithms: the ring artifact reduction algorithm by Postnov and Sijbers [2], the vessel enhancement fitler (also known as vesselness filter) by Frangi *et al.* [3]. The centerline extraction procedure is inspired by the ideas of the Canny edge detector [31]. Zhong *et al.* [12] described in details and implemented these parts of our algorithm. The curvature regularization is based on the curvature estimation formula proposed by Olsson *et al.* [32]. The general thin structure estimation model using this approximation formula is formulated by Marin *et al.* [20].

This thesis presents two distinct contributions. First, we provide an algorithm where the ill-posed vessel centerline extraction problem is solved by optimizing an objective function combining data alignment and curvature-based smoothness prior. We propose an efficient implementation with a GPU as a target computing platform. Our CUDA-accelerated algorithm exploits sparsity of the optimization problem and allows to estimate vessel centerline from ultra high resolution volumes (see Figure 1.1) in reasonable time. Second, we developed a validation mechanism based on a new formulation of a tree matching problem minimizing Fréchet distance. Using this optimal matching we compare three methods for centerline estimation from 3D synthetic vascular volume: accurate fast marching algorithm for skeleton-based esti-

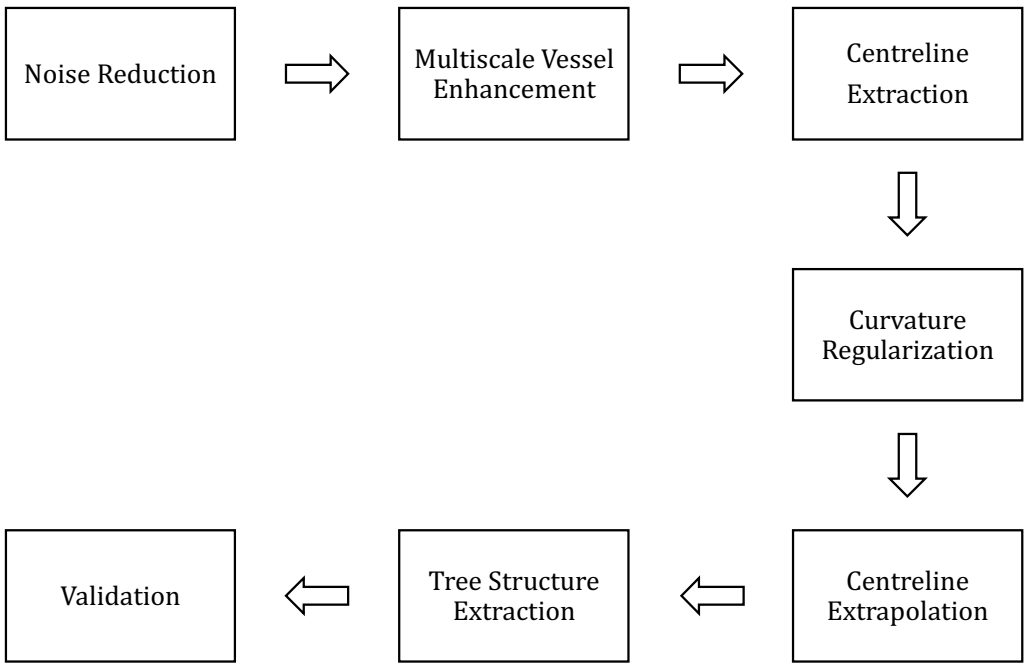


Figure 1.8: Vascular tree centerline extraction algorithm.

mation[2], centerline estimation method [12] inspired by the ideas of Canny-edge detector, and our centerline estimation method with curvature-based regularization.

1.4 Thesis Structure

The remainder of the thesis is organized as follows. Chapter 2 outlines the parts of our vessel centerline estimation framework. Chapter 3 addresses challenging problem of the curvature regularization. We discuss implementation details, using optimization algorithms and target computing platform. Chapter 4 presents validation mechanism. We transform validation problem to matching problem and solve it by applying an efficient algorithm. We demonstrate promising results of experimental evaluation based on synthetic generation of vascular volumes [33, 34]. Chapter 5 concludes the thesis and introduce possible directions of future work.

[2] http://www.mathworks.com/matlabcentral/fileexchange/24531-accurate-fast-marching

Chapter 2

Overview of Vessel Centerline Estimation

This chapter outlines the basic elements of our vessel extraction approach (see Figure 1.8). The ring artifacts reduction technique [2] was described in Subsection 1.2.1. Vessel enhancement filtering [3], a standard mechanism to reduce noise and address low contrast in volumes, is discussed in Section 2.1. Standard skeleton-based centerline approaches and their problems in the context of the partial volume effect are discussed in Section 2.2. Our curvature-based regularization framework for vessel centerline estimation (see Section 2.3) avoids pre-segmentation. It directly uses the output of the vesselness filter (Section 2.1). Besides the scalar measure of vesselness, our algorithm also uses its vessel direction estimate for initialization. Then, we compare results of our method and results obtained by skeleton-based estimation algorithm. More detailed validation is presented in Chapter 4.

2.1 Vessel Enhancement Filtering

This section describes multi-scale vessel enhancement algorithm developed by Frangi, *et al.* [3]. The objective of applying this algorithm is threefold: to remove random noise in an image data, to increase contrast between voxels belonging to blood vessels and nonvascular structures (e.g. muscle tissue), and to estimate directions along vessels centerlines.

Consider an intensity function $I(p)$ giving the intensity at each voxel p of an image. Approximate its value by Taylor expansion up to second order in the neighborhood of $p = (x, y, z)$

$$I(p + \delta p) \approx I(p) + \delta p^T \nabla I + \delta p^T H \delta p, \tag{2.1}$$

where ∇I and H are the gradient vector and the Hessian matrix of I computed at p, respectively.

The Hessian H describes the second-order intensity variations around p:

$$H = \begin{vmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial x \partial z} \\ \frac{\partial^2 I}{\partial y \partial x} & \frac{\partial^2 I}{\partial y^2} & \frac{\partial^2 I}{\partial y \partial z} \\ \frac{\partial^2 I}{\partial z \partial x} & \frac{\partial^2 I}{\partial z \partial y} & \frac{\partial^2 I}{\partial z^2} \end{vmatrix}. \tag{2.2}$$

Consider the eigenvalue/eigenvector decomposition of the Hessian H:

$$H\phi_k = \lambda_k \phi_k, k = 1, 2, 3 \tag{2.3}$$

where $\phi_1,\ \phi_2,\ \phi_3$ are its eigenvectors and $\lambda_1,\ \lambda_2,\ \lambda_3$ are the corresponding eigenvalues. Without loss of generality assume that eigenvalues are ordered: $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|$. The eigenvector ϕ_3 represents the direction along which the second derivative is maximum. Similarly, ϕ_1 represents the minimum second-derivative value direction.

Three following second-order local structures can be identified by analyzing eigenvalues: a plate-like structure ($|\lambda_1| \approx 0,\ |\lambda_1| \approx |\lambda_2|,\ |\lambda_2| \ll \lambda_3$), a line-like structure ($|\lambda_1| \approx 0,\ |\lambda_1| \ll |\lambda_2|,\ |\lambda_2| \approx |\lambda_3|$) and a blob-like structure ($|\lambda_1| \approx |\lambda_2| \approx |\lambda_3|$).

Frangi, *et al.* [3] defined *vesselness measure*

$$V(p) = \begin{cases} 0 & \text{if } \lambda_2 > 0 \text{ or } \lambda_3 > 0 \\ \Big(1 - \exp\Big(-\frac{R_A^2}{2\alpha^2}\Big)\Big) \exp\Big(-\frac{R_B^2}{2\beta^2}\Big) \Big(1 - \exp\Big(-\frac{S^2}{2c^2}\Big)\Big) & \text{otherwise} \end{cases} \tag{2.4}$$

based on the composition of the following three features:

$$\begin{aligned} R_B &= \frac{|\lambda_1|}{\sqrt{|\lambda_2 \lambda_3|}}, \\ R_A &= \frac{|\lambda_2|}{|\lambda_3|} \text{ and} \\ S &= \sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}. \end{aligned} \tag{2.5}$$

R_B distinguishes between a blob-like structure ($R_B \approx 1$) and a line-like (e.g. vessel) or a plate-like structure ($R_B \approx 0$). R_A distinguishes between a line-like ($R_A \approx 1$) and a plate-like ($R_A \approx 0$) structures. The Frobenius norm S has a maximal response at high contrast region and low response at the background. Hence the *vesselness function* in Equation (2.4) discriminates between voxels belonging to blood vessels (characterized by small $|\lambda_1|$ and large $|\lambda_2|,\ |\lambda_3|$) and other voxels. The eigenvector ϕ_1 indicates the direction along the vessel (minimum intensity variation). α, β, c are parameters which control the sensitivity of the vesselness measure to features R_A, R_B, S, respectively.

In practice, vesselness measure is computed at different scales for a *scale-space representation* of an image:

$$I(p, \sigma) = G(p, \sigma) * I(p), \tag{2.6}$$

where symbol $*$ denotes convolution over x, y, z and

$$G(p, \sigma) = \frac{1}{\sqrt{(2\pi\sigma^2)^3}} \exp\left(- \frac{x^2 + y^2 + z^2}{2\sigma^2} \right) \tag{2.7}$$

is the three-dimensional Gaussian kernel. Then, the maximum response,

$$V(p) = \max_{\sigma_{min} \leq \sigma \leq \sigma_{max}} V(p, \sigma), \tag{2.8}$$

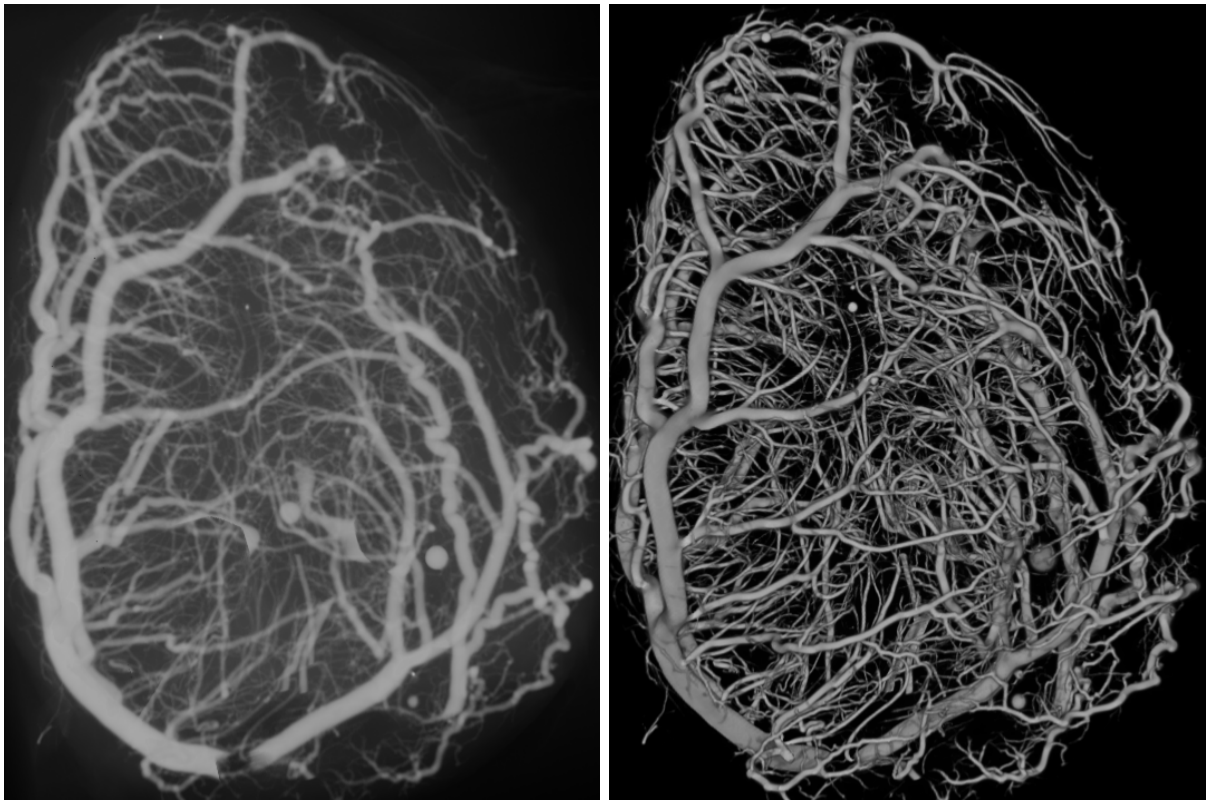
is chosen to be a final estimate of vesselness measure at voxel p. The scale σ with a maximum response, $V(p, \sigma)$, approximates the size of the vessel. Note that σ_{min} and σ_{max} have to be chosen to cover the expected range of vessel radiuses.

Figures 2.1 and 2.2 illustrate the result of applying vessel enhancement filtering to a CT image. Note that while this method is very effective, it is still not enough to completely disambiguate the vessels from the background. In particular, the filter does not work well on near-capillary vessels with severe partial volume problems (see Figures 2.2c and 2.2d).

2.2 Standard Centerline Extraction

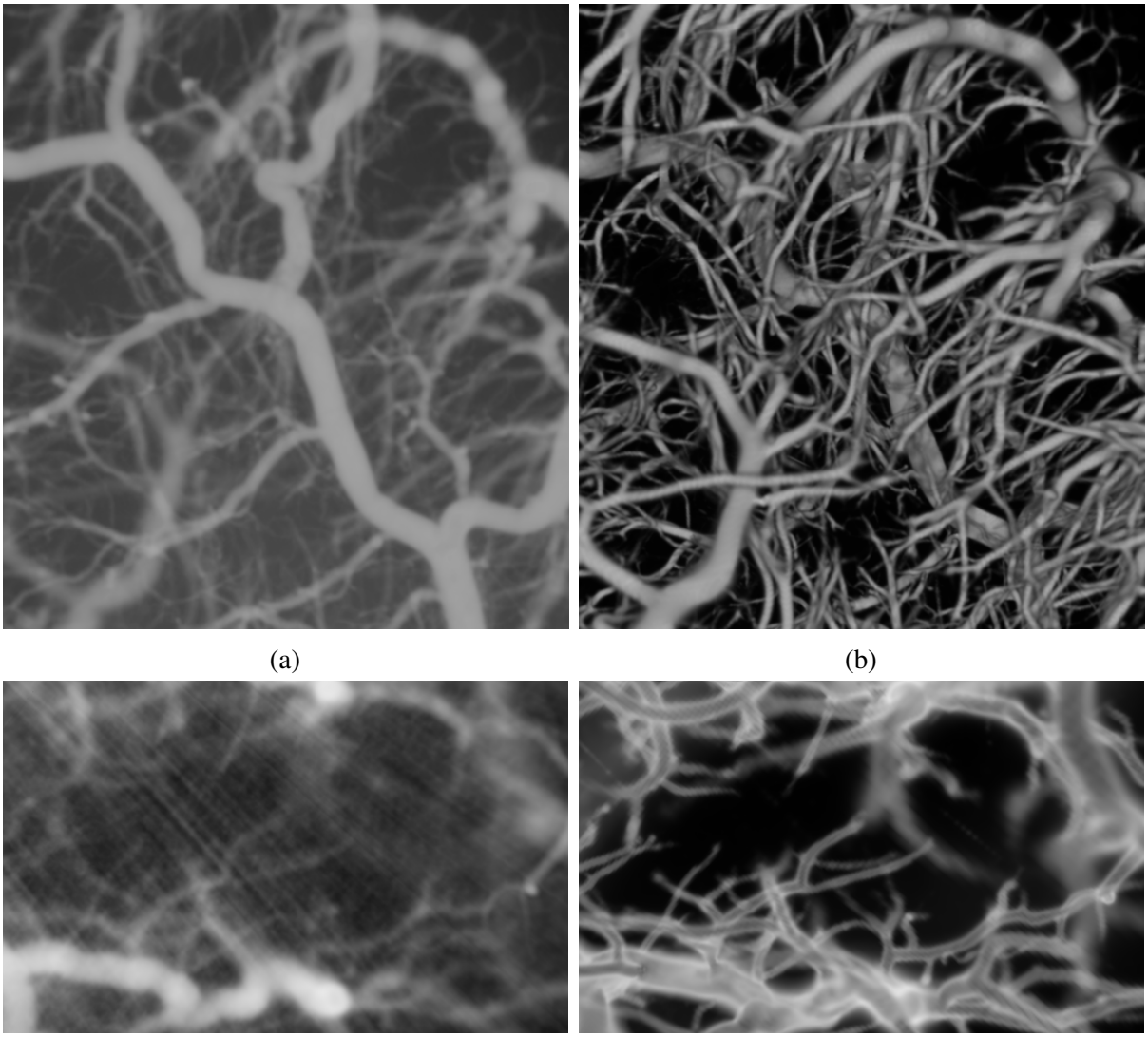
One of the most popular approaches for centerline estimation is medial representation of objects, also known as *skeleton* or *medial axis* [35, 21, 8, 9, 10]. Formally, a skeleton is defined as the set of points that have more than one closest point on the boundary [35]. Generally, a skeleton is not robust to irregularities of object's boundary (see Figure 2.3), thereby additional regularization is required. Moreover, these methods require binary segmentation of the image. Straightforward methods of binary segmentation (e.g. thresholding) produce unreliable results: often thin vessels disappear, while noise is labeled as a part of the vessel (see Figure 2.7c). Figure 2.7 illustrates the result of skeleton extraction from an image enhanced by Frangi, *et al.* [3] filter.

In contrast, our approach directly estimates the smooth centerline and thus does not require intermediate binary mask processing.



(a) (b)

Figure 2.1: Vessel enhancement filtering. (a) shows the 3D rendered original volume while (b) shows the volume after applying the multi-scale vessel enhancement algorithm developed by Frangi *et al.* [3].



(a) (b)

(c) (d)

Figure 2.2: Vessel enhancement filtering. (a) and (c) show the 3D rendered volume at different scales while (b) and (d) show the volume after applying the multi-scale vessel enhancement algorithm developed by Frangi *et al.* [3]. Note that the thin vessels and the background in (d) are indistinguishable by intensity values alone.

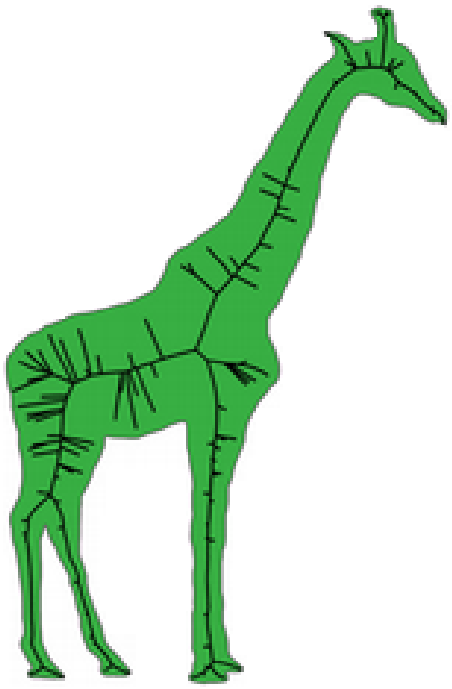


Figure 2.3: Example demonstrates instability of non-regularized medial axis to small disturbances of the object boundary.[1]

2.3 Our Curvature Regularization Framework

In this section we are concerned with curvature-based regularization of the centerline estimation problem. Due to unknown topology and the presence of noise and the outliers, the problem of centerline estimation is *ill-posed*. We impose a smoothness prior on model parameters defining centerline in order to solve this problem.

Zhong [12] implemented curvature extraction algorirthm that adopted two important ideas from Canny edge detector [31]. First, they applied the non-maximum suppression procedure, see Figures 2.4a and 2.4b, to the vector field of the estimated vessel directions (determined by eigenvector ϕ_1 corresponding to eigenvalue λ_1 with the smallest absolute value in Equation (2.3)) obtained by the vessel enhancement filter described in Section 2.1. Then, they used a double threshold to extract points that are most likely located near the centerline (see Figures 2.4c, 2.4d and 2.4e). We use the result of these two steps as an input for our regularization framework.

Let L denote some abstract hypothetical centerline. We make two assumptions: first, that measurements $\{\tilde{p}\}$ estimates the position of the centerline L with some accuracy and second, that the centerline L is *"smooth"*. Figure 2.5a illustrates an example of such centerline L along

[1] `http://www.agg.ethz.ch/research/medial_axis`

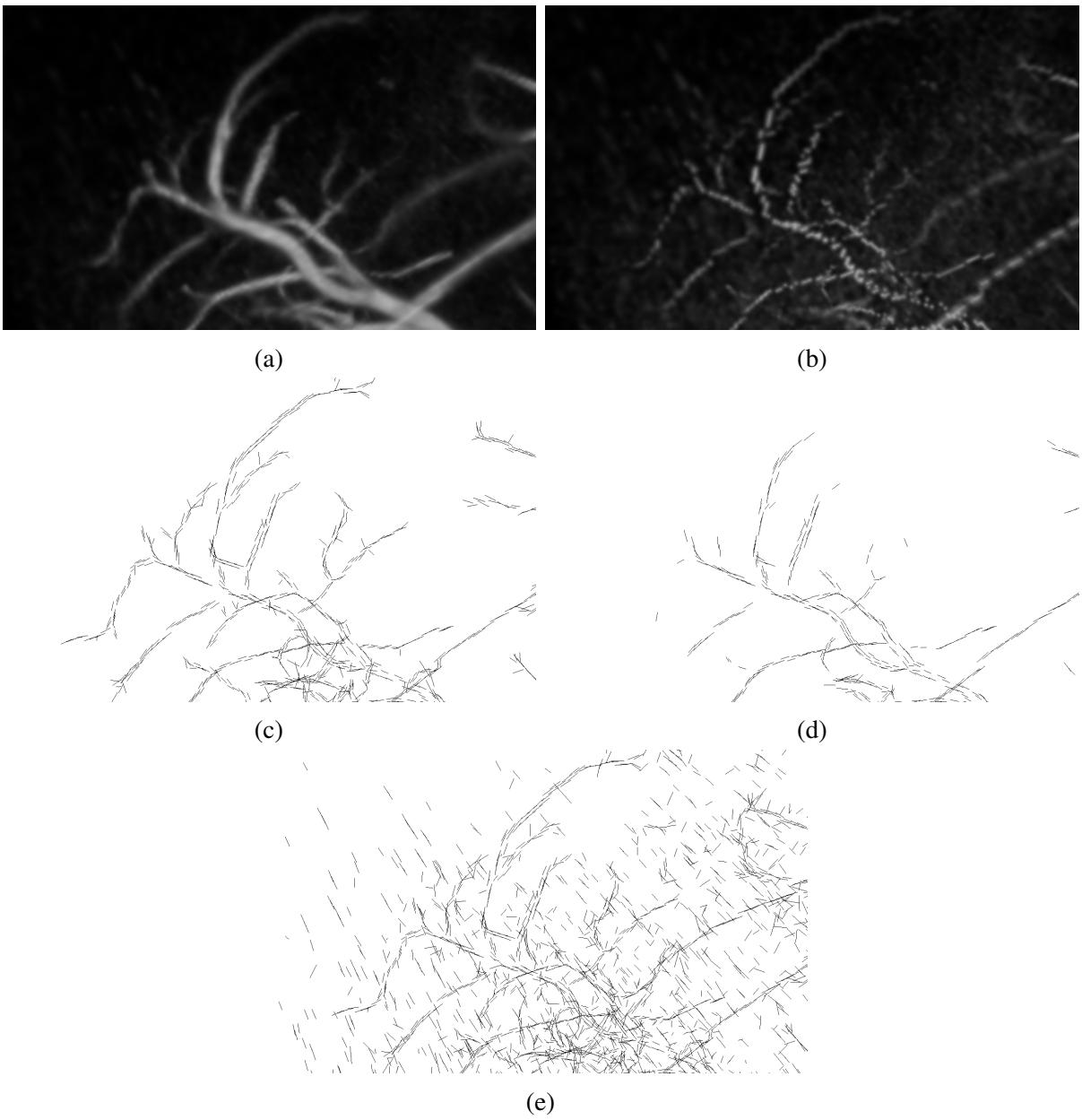


Figure 2.4: Our method adopted two ideas of Canny-edge detector. First, non-maximum suppression. (a) The input volume of the method. (b) Result of the non-maximum suppression procedure applied for the volume in (a). Second, double thresholding, also known as hysteresis. (c) Result of double thresholding are more robust that single-threshold approach. (d) A threshold limit set too high can reject some parts of vessel tree. (e) On other hand, threshold limit set too low will falsely identify noise as a part of vessel.

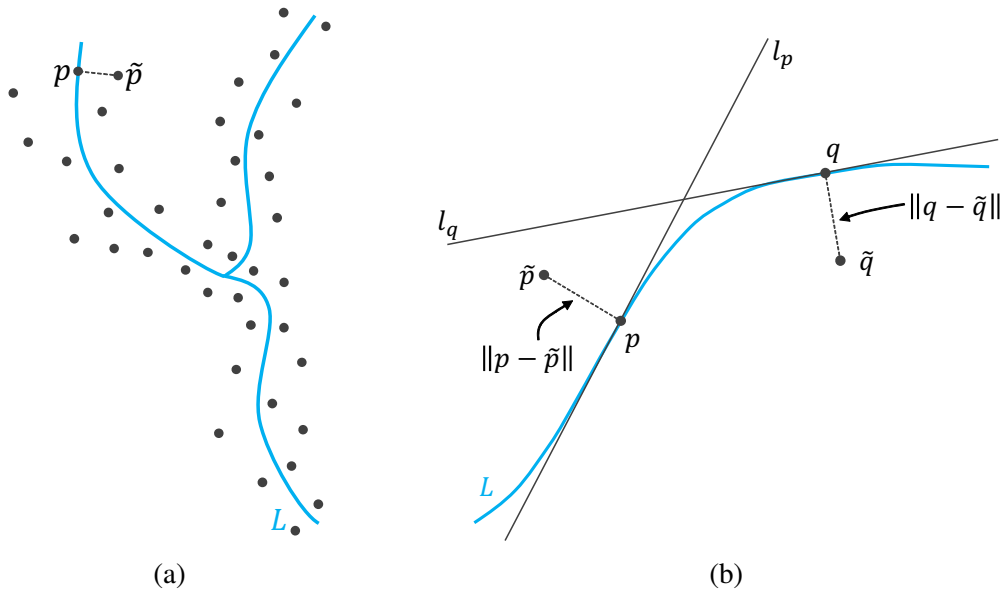


Figure 2.5: Centerline definition. (a) illustrates an example of the noisy measurements $\{\tilde{p}\}$ along with original centerline L. (b) shows displacements between points p and q on centerline L and their noisy measurements $\tilde{p}$ and $\tilde{q}$.

with measurements $\{\tilde{p}\}$.

Our objective is to find the *"smooth"* centerline L that *lies in the neighborhood* of its noisy measurements $\{\tilde{p}\}$. In other words, we incorporate *smoothness prior* knowledge with noisy *measurements data* and look for the centerline L that satisfies both. This can be represented formally by introducing two cost functions: $E_{smooth}(L)$ and $E_{data}(L)$. Function $E_{smooth}(L)$ penalizes for L being non-straight, while $E_{data}(L)$ is a cost for the displacement with respect to measurements $\{\tilde{p}\}$. Thus, we look for centerline L that minimizes the following energy function

$$E(L) = E_{smooth}(L) + \alpha E_{data}(L) \tag{2.9}$$

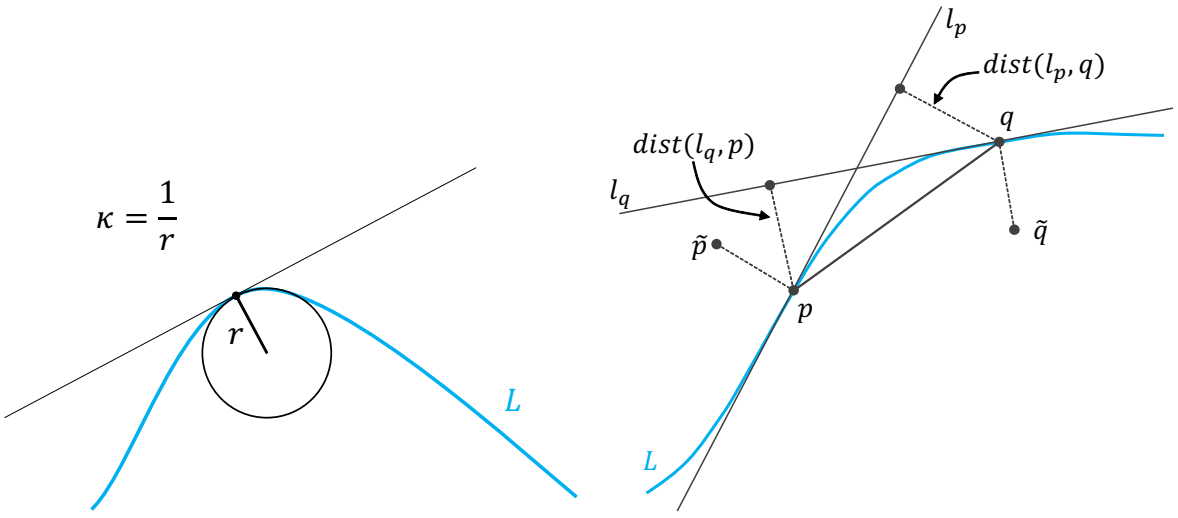
where $\alpha > 0$ controls trade-off between smoothness prior term $E_{smooth}(L)$ and data consistency term $E_{data}(L)$.

Let p denote orthogonal projection of its measurement $\tilde{p}$ onto the centerline L and l_p denote the tangent direction of the centerline L at p (see Figure 2.5b). Thus, we parameterize the centerline L by introducing its tangent directions $\{l_p\}$. We define a data consistency term $E_{data}(L)$ as follows:

$$E_{data}(L) = \sum_{p} \frac{1}{\sigma_p^2} \|p - \tilde{p}\|^2, \tag{2.10}$$

where σ_p is level of noise.

Curvature is a natural measure of *regularity* or *non-linearity* of a curve. Geometrically, the



(a) (b)

Figure 2.6: Curvature approximation. (a) demonstrates a definition of curvature as reciprocal of *osculating circle* radius. (b) illustrates pairwise interaction between p and q, their noisy measurements $\tilde{p}$ and $\tilde{q}$, and distances between points p, q and tangent lines l_q, l_p, respectively.

curvature of a circle is the reciprocal of its radius. The curvature of a straight line is defined to be zero. The curvature of any sufficiently smooth curve L at a point can be defined as the curvature of the circle that most closely approximates L at this point (see Figure 2.6a).

We approximate the smoothness term, $E_{smooth}(L)$, by an equation proposed by Olsson *et al.* [32, 36]. They approximate absolute curvature as follows:

$$\kappa(l_p, l_q) = \frac{dist(l_q, p) + dist(l_p, q)}{\|p - q\|}, \tag{2.11}$$

where $dist(l_p, q)$ denotes the distance between point q and its orthogonal projection onto the tangent line l_p (see Figure 2.6b). Similarly, the following approximates the squared curvature:

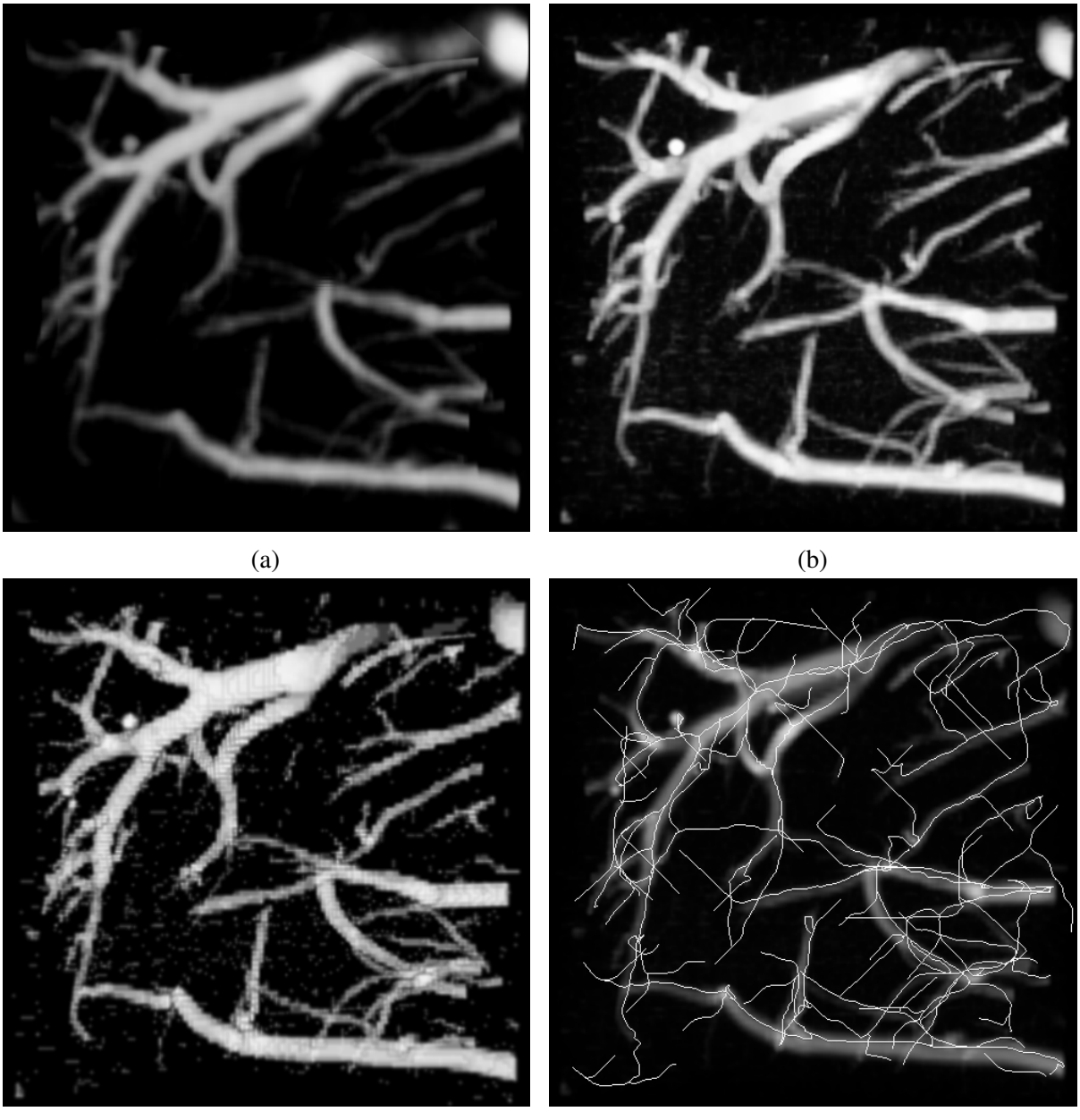
$$\kappa^2(l_p, l_q) = \frac{dist^2(l_q, p) + dist^2(l_p, q)}{\|p - q\|^2}. \tag{2.12}$$

We estimate the centerline tangents $\{l_p\}$ from noisy measurements $\{\tilde{p}\}$ by minimizing the following objective function

$$\sum_p \sum_{q \in \mathcal{N}(p)} \frac{dist^2(q, l_p)}{\|p - q\|^2} + \sum_p \frac{\alpha}{\sigma_p^2} \|\tilde{p} - p\|^2, \tag{2.13}$$

where $\mathcal{N}(p)$ denotes the neighborhood of p and σ_p denotes the radius of vessel. In other words, we assign tangent line l_p to each noisy measurement $\tilde{p}$ that minimizes the squared distances

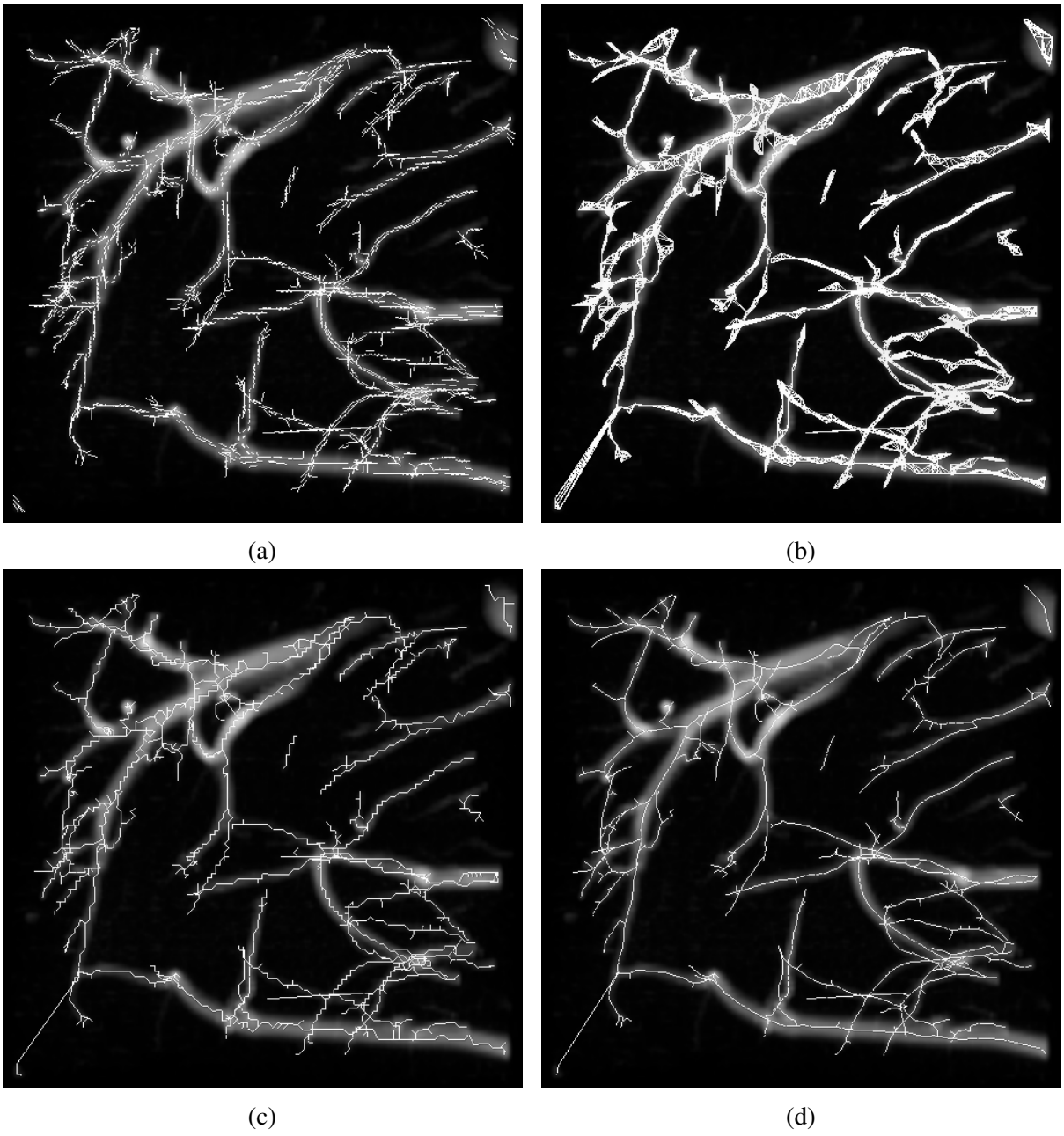
to the point p and the non-smoothness of the centerline defined by the curvature between neighboring points. Note that σ_p in Equation (2.10) is defined as the noise level and here we assume that this is proportional to the scale of vessel. Figure 2.8 illustrates the result of centerline regularization.



(a) (b)

(c) (d)

Figure 2.7: Application of a skeleton-based algorithm for vascular tree extraction. (a) Original volume. (b) Result of vessel enhancement filtering. (c) Binary segmentation (thresholding) of (b). (d) Skeleton extracted from (c) by an accurate fast marching algorithm [8, 9, 10] implementation of Kroon.



(a) (b)

(c) (d)

Figure 2.8: Curvature-based centreline extraction. (a) The vector field of vessel directions produced by vessel enhancement filtering. (b) The neighborhood graph defined by $\mathcal{N}(p)$ in Equation (2.13). The graph reflects our assumption about centreline topology (we assume k-nearest neighbors connectivity). (c) The minimum spanning tree extracted directly from the graph in (b). (d) The result of centreline regularization (minimizing the objective function in Equation (2.13)) followed by minimum spanning tree extraction.

Chapter 3

Fast Curvature Regularization

In this chapter we review optimization algorithms addressing a large scale non-linear least squares problem of the form $\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} f_i^2(x)$ for which the Jacobian is sparse. Problems of this type appear in the context of curvatude-based regularization. The following two algorithms are discussed in Section 3.2: an inexact Levenberg-Marquardt algorithm [37] and an algorithm called LSQR by Paige and Saunders for sparse linear least squares [38]. We also talk about sparse matrix storage methods and automatic differentiation [39], a method for numerical evaluation of the derivatives in Section 3.3. Then, we present our GPU implementation of the non-linear least squares solver for the parallel computing platform CUDA in Section 3.4.

3.1 Problem Overview

The curvature-based regularization problem is formulated as an optimization problem of the following form:

$$\min_{l_p} \sum_{p} \sum_{q\in\mathcal{N}(p)} V^2(l_p, l_q) + \sum_{p} D^2(l_p), \tag{3.1}$$

where:

$$D^2(l_p) = \frac{1}{\sigma_p^2}\|\tilde{p} - p\|^2, \tag{3.2}$$

is unary cost function penalyzing the discrepancy between the measurement $\tilde{p}$ and its *denoised* version p and

$$V^2(l_p, l_q) = \frac{dist^2(q, l_p)}{\|p - q\|^2} \tag{3.3}$$

is the pairwise cost function penalyzing the curvature between tangent lines l_p and l_q at neighboring points p and q.

We parameterize the tangent line l_p by introducing two distinct points $s_p, t_p \in \mathbb{R}^3$ on that

line. Thus, we can describe any point $r_p \in l_p$ by parametric equation:

$$r_p(\alpha) = s_p - \alpha(s_p - t_p), \tag{3.4}$$

where $\alpha \in \mathbb{R}$ is a parameter. For example, the orthogonal projection p of $\tilde{p}$ onto the line l_p is

$$p = s_p - \alpha^{\perp}(s_p - t_p), \quad \alpha^{\perp} = \frac{(s_p - \tilde{p})(s_p - t_p)}{\|s_p - t_p\|^2}. \tag{3.5}$$

One can notice that Equation (3.1) is an unconstrained non-linear least squares problem of the form:

$$\min_x F(x) = \|f(x)\|_2^2 = f(x)^T f(x), \tag{3.6}$$

where variable x concatenates coordinates of points s_p and t_p

$$x = \begin{bmatrix} \vdots \\ s_p \\ t_p \\ \vdots \\ s_q \\ t_q \\ \vdots \end{bmatrix} \in \mathbb{R}^n, \qquad f(x) = \begin{bmatrix} \vdots \\ D(l_p) \\ \vdots \\ D(l_q) \\ \vdots \\ V(l_p, l_q) \\ \vdots \end{bmatrix} \in \mathbb{R}^m. \tag{3.7}$$

There is a variety of numerical methods that have been developed to solve Equation (3.6). Note that our application yields problems with from thousands to millions of variables. Hence computational and storage costs become a critical selection criteria of an optimization approach.

3.2 Outline of Optimization Algorithms

In this section we are concerned with the numerical algorithms for solving non-linear least squares of the form in Equation (3.6). According to Nocedal and Wright [11], all algorithms for solving the non-linear least squares can be classified as:

1. **Line Search:** Line search algorithms at each iteration generate a search direction p_k, and then determine step size α_k (how far one should move along that direction). The next iteration is given by formula

$$x_{k+1} = x_k + \alpha_k p_k. \tag{3.8}$$

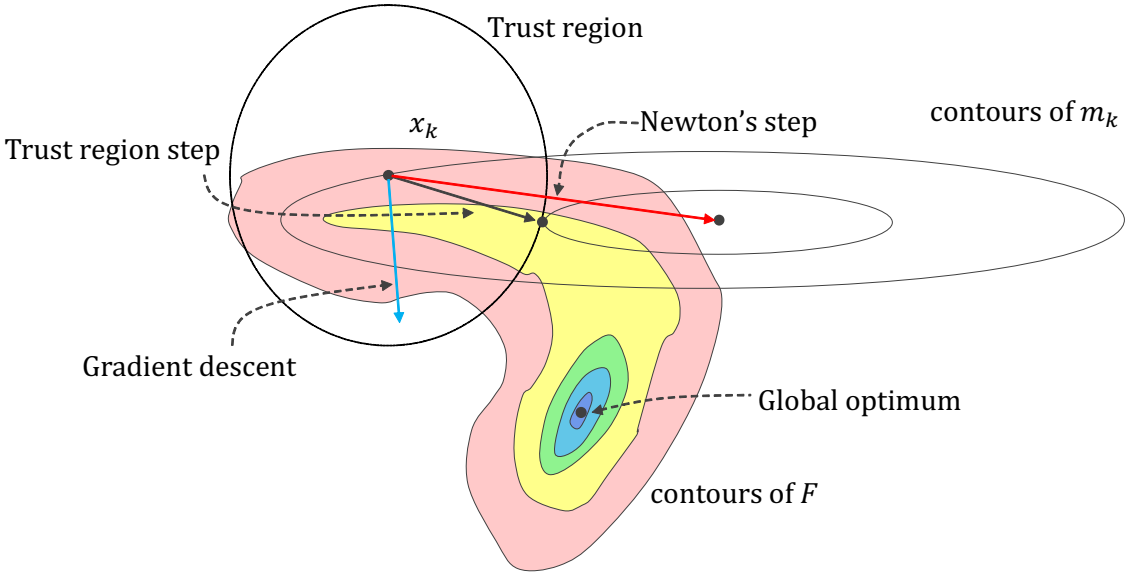


Figure 3.1: Trust region and line search optimization strategies [11]. Note that neither Newton's step (red arrow) nor gradient descent (blue arrow) go in direction of global optimum. Quadratic function m_k approximates the objective function F accurately only in a small neighborhood of x_k where the approximation m_k can be *trusted* (so-called *trust region*). In other words, trust region is defined as a region where the decreasing of the model function m_k implies the sufficient decreasing of the objective function F.

One straightforward choice for a search direction is $p_k = -\nabla F(x_k)$, which is the negative of the gradient.

2. **Trust Region:** Trust region algorithms generate steps with the help of another so-called model function m_k that approximates the objective function $F(x)$ in the neighborhood of the current iterate x_k. They first define a region around the current iterate x_k within which they *trust* the model function m_k to be an adequate approximation of the objective function, and then choose the step to be an approximate minimizer of the function m_k in this region. In effect, they choose the direction and length of the step, simultaneously. If a step does not lead to a sufficient decreasing of the objective function value, the trust region is contracted and they find a new minimizer m_k.

Figure 3.1 illustrates difference between trust-region and line search methods.

3.2.1 Levenberg-Marquardt Algorithm

Levenberg-Marquardt algorithm [40, 41] is an algorithm for solving non-linear least squares problems. The algorithm adopts the trust region approach and approximates the objective

function:

$$F(x) = f^T(x)f(x) \tag{3.9}$$

in Equation (3.6) with the model function:

$$m_k(x) = \tilde{f}_k^T(x)\tilde{f}_k(x), \tag{3.10}$$

where:

$$\tilde{f}_k(x) = f(x_k) + J^T(x_k)(x - x_k), \tag{3.11}$$

is an approximation of $f(x)$ by its first-order Taylor expansion in the neighborhood of x_k. $J(x_k)$ denotes the Jacobian matrix of $f(x)$. Hence $m_k(x)$ is defined by:

$$m_k(x_k + \delta x) = f^T(x_k)f(x_k) + 2\delta x^T J^T(x_k)f(x_k) + \delta x^T J^T(x_k)J(x_k)\delta x. \tag{3.12}$$

Note that although $m_k(x)$ is a quadratic approximation of $F(x)$, this is not the same as the second-order Taylor expansion of $F(x)$:

$$F(x_x + \delta x) \approx f^T(x_k)f(x_k) + 2\delta x^T J^T(x_k)f(x_k) + \delta x^T H(x_k)\delta x. \tag{3.13}$$

where $H(x_k)$ denotes the Hessian matrix of $F(x)$ computed at x_k. The computation of $H(x_k)$ can be expensive for higher dimensional problems, thereby the Hessian matrix $H(x_k)$ is approximated in Equation (3.12) by the Jacobian matrix square $J^T(x_k)J(x_k)$.

In order to find the minimum of the model function m_k

$$\min_{\delta x} m_k(x_k + \delta x) \tag{3.14}$$

we compute the gradient $\nabla m_k(x_k + \delta x)$ first as:

$$\nabla m_k(x_k + \delta x) = 2J^T(x_k)f(x_k) + 2J^T(x_k)J(x_k)\delta x \tag{3.15}$$

and then, setting the latter to zero gives us the system of linear equations:

$$J^T(x_k)J(x_k)\delta x = -J^T(x_k)f(x_k). \tag{3.16}$$

Levenberg [40] proposed using a damping parameter $\lambda_k > 0$ that implicitly controls the size of trust region in Equation (3.14) at each iteration of the method:

$$\min_{\delta x} m_k(x_k + \delta x) + \lambda_k ||\delta x||_2^2 \tag{3.17}$$

thereby replacing the system of equations in Equation (3.16) with the following damped version

$$(J^T(x_k)J(x_k) + \lambda_k I)\delta x = -J^T(x_k)f(x_k). \tag{3.18}$$

There are various strategies describing how to choose the parameter λ_k at each iteration [41].

The standard implementation of this algorithm computes the Jacobian square $J^T(x_k)J(x_k)$ and, then, solves the linearized subproblem given in Equation (3.18) at each iteration. This can be prohibitive for problems involving real CT data of large size. For example, when we solve the regularization problem for our data, the matrix $J^T(x_k)J(x_k)$ in Equation (3.18) has size of million rows by million columns. Furthermore, we show that the Jacobian $J(x)$ for Equation (3.1) is a sparse matrix. We conclude that there is a need for an optimization method that takes into consideration the sparsity and the large-scale of our problem.

3.2.2 Inexact Levenberg-Marquardt Algorithm

We review the inexact Levenberg-Marquardt algorithm [37], a method based on the Levenberg-Marquardt strategy for solving large sparse non-linear least squares problems. The algorithm internally uses the iterative method LSQR [38] for solving linear least squares in Equation (3.17), that can be explicitly written in the following way:

$$\min_{\delta x} ||J(x_k)\delta x + f(x_k)||_2^2 + \lambda_k ||\delta x||_2^2. \tag{3.19}$$

The inexact Levenberg-Marquardt algorithm does not compute $J^T(x_k)J(x_k)$ at all. In contrast, LSQR iteratively solves the linear least squares problem given in Equation (3.19) using only the operations of the matrix-vector multiplications of the form $J^T u$ and Jv where u and v are vectors. These operations are significantly cheaper (in meaning of computational and memory complexity) than the operations of matrix-matrix multiplication of the form $J^T J$.

Wright and Holt [37] have shown that to achieve a substantial reduction in the sum of squares and avoid extra computational effort one can apply only a few iterations of LSQR to Equation (3.18).

A simplified version of algorithm [37] is listed in Figure 3.2. Parameters π_1, π_2, E, D and $\lambda_{\min}$ control the choice of damping parameter λ_k. LSQR minimizes Equation (3.19), thereby predicting the possible step $x_{k+1} = x_k + \delta x$. Wright and Holt [37] propose to make the decision about whether the step is accepted or not by the calculating of the ratio:

$$\rho = \frac{\text{actual reduction}}{\text{predicted reduction}} = \frac{F(x_k) - F(x_k + \delta x)}{F(x_k) - m_k(x_k + \delta x) - \lambda_k ||\delta x||_2^2}. \tag{3.20}$$

Input: x_0
Parameters: $0 < \pi_1 < \pi_2 < 1, E > 1, D < 1, \lambda_{\min} > 0$

evaluate $J(x_0)$
$k \leftarrow 0$
repeat
  $\delta x \leftarrow$ LSQR minimizes $\|J(x_k)y + f(x_k)\|_2^2 + \lambda_k\|\delta x\|_2^2$
  $\rho \leftarrow (F(x_k) - F(x_k + \delta x))(F(x_k) - m_k(x_k + \delta x) - \lambda_k\|\delta x\|_2^2)^{-1}$
  if $\rho < \pi_1$ **then** *step does not sufficiently decrease the objective function value, the trust region is contracted*
    if $\lambda_k = 0$ **then**
      $\lambda_k \leftarrow \lambda_{\min}$
    else
      $\lambda_k \leftarrow E\lambda_k$
    end
  else *an algorithm "trusts" the model function and the step is accepted*
    $x_{k+1} \leftarrow x_k + \delta x$
    evaluate $J(x_{k+1})$
    if $\rho > \pi_2$ **then** *the trust region is expanded*
      $\lambda_{k+1} \leftarrow D\lambda_k$
      if $\lambda_{k+1} < \lambda_{\min}$ **then** *the occurrence of Gauss-Newtow step*
        $\lambda_{k+1} \leftarrow 0$
      end
    end
    $k \leftarrow k + 1$
  end
until *convergence is achieved to the desired tolerance*;

Result: x_k

Figure 3.2: An inexact Levenberg-Marquardt algorithm.

Note that in case $\lambda_k = 0$ the step x_{k+1} of Levenberg-Marquardt algorithm is identical to the step of Gauss-Newton algorithm, an algorithm for solving non-damped non-linear least squares as given in Equation (3.14).

Note also that most of the computational effort in each iteration lies in solving the linear Equation (3.18) with LSQR and evaluating the Jacobian matrix $J(x)$.

3.2.3 LSQR: Sparse Linear Equations and Least Squares Problems

LSQR [38] is an iterative method developed by Paige and Saunders to find a solution to the following problems:

1. **System of linear equations:**

$$Ax = b$$

2. **Linear least squares:**

$$\min_x ||Ax - b||_2^2$$

3. **Damped least squares:**

$$\min_x ||Ax - b||_2^2 + \lambda ||x||_2^2$$

where A is a real matrix with m rows and n columns and $b \in \mathbb{R}^m$ is a real vector.

The method is based on the bidiagonalization algorithm of Golub and Kahan [42]. The main idea is to generate a sequence of solutions $\{x_k\}$ such that the residual norms $\{||b-Ax_k||_2\}$ are monotonically decreasing. For the damped least squares problem this is equivalent to applying the conjugate gradient method to the following equation:

$$(A^T A + \lambda I)x = A^T b. \tag{3.21}$$

However, LSQR has been shown to be more numerically reliable than other numerical methods in the context of sparse and large systems of linear equations. We refer the reader interested in topic to Paige and Saunders [38] for a more detailed description of the algorithm.

Note that the matrix A is used in LSQR only to compute products of the form Av and $A^T u$ for various vectors $v \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$. These computations are much cheaper in the sense of computational and storage costs than those required by a numerical method solving Equation (3.18).

3.3 Jacobian Matrix: Storage and Evaluation

If the Jacobian matrix $J(x)$ is sparse, the large-scale linear least squares problem of the form in Equation (3.18) can be efficiently solved if the zero elements are not stored in a physical computer memory. There are many methods for storage of the sparse matrices. We review CSR (Compressed Row Storage) in Subsection 3.3.1, a method most suited for storage the Jacobian $J(x)$ that arises in the context of the problem given in Equation (3.1).

There are different ways to obtain derivatives. For example, one can go back to the function equation and find derivatives mechanically. Despite that it works perfectly for simple functions the process becomes complex and error-prone for functions of many arguments. Furthermore, if the equation defining the objective function changes we have to repeat the process from the beginning. Alternatively, numerical values of derivatives can be approximated in form of

partial differences. However, that method can introduce round-off errors in the discretization process. Hence we look at the more practical ways in Subsection 3.3.2.

3.3.1 Compressed Row Storage

Consider the Jacobian for the objective function in a non-linear least squares problem of the form given by Equation (3.1), which can be represented in the following symbolic form:

$$J(\dots l_p \dots l_q \dots) = 2 \begin{vmatrix} D(l_p)\frac{\partial D(l_p)}{\partial l_p} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & D(l_q)\frac{\partial D(l_q)}{\partial l_q} \\ \vdots & & \vdots \\ V(l_p, l_q)\frac{\partial V(l_p, l_q)}{\partial l_p} & \dots & V(l_p, l_q)\frac{\partial V(l_p, l_q)}{\partial l_q} \end{vmatrix}. \tag{3.22}$$

Note that since Equation (3.1) at maximum contains only pairwise interactions between tangent lines l_p and l_q, each row of the Jacobian matrix in Equation (3.22) depends only on few variables (a row corresponding to pairwise term $V(l_p, l_q)$ depends on 12 unknown variable, since both tangent lines l_p and l_q are parameterized by two three-dimensional points s_p, t_p and s_q, t_q). Consequently, most of elements in the matrix are zeros. There are a number of common storage formats used for sparse matrices, but most of them employ the same basic technique[1]: store all nonzero elements of the matrix into an array data structure and provide auxiliary data structures to describe the position of the non-zero elements in the sparse matrix.

The CRS (Compressed Row Storage) format is represented by three arrays `val`, `col_ind`, `row_ptr`. Array `val` contains the non-zero elements of a sparse matrix obtained by traversing across each row in order. The i-th element of array `col_ind` stores the column index of the i-th item in array `val`. The k-th element of array `row_ptr` points to the item in array `val` that starts a row k.

For example, consider the following matrix:

$$\begin{vmatrix} 0 & 0 & a_{13} & a_{14} & 0 \\ a_{21} & 0 & 0 & 0 & a_{25} \\ 0 & a_{32} & 0 & a_{34} & 0 \\ a_{41} & 0 & 0 & 0 & 0 \end{vmatrix}.$$

[1] `https://software.intel.com/en-us/node/471374`

The arrays `val`, `col_ind`, `row_ptr` for this matrix are given below

i	1	2	3	4	5	6	7
val	a_{13}	a_{14}	a_{21}	a_{25}	a_{32}	a_{34}	a_{41}
col_ind	3	4	1	5	2	4	1

k	1	2	3	4	5
row_ptr	1	3	5	7	8

By convention, we add an extra item `row_ptr[5]` in the array `row_ptr` that contains the value `length(val) + 1`.

3.3.2 Automatic Differentiation

In this section we are concerned with a numerical algorithm for the evaluation of derivatives. Since

$$\frac{df}{dx} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{3.23}$$

one can easily approximates derivatives with some form of finite difference

$$\frac{df}{dx} = \frac{f(x+h) - f(x)}{h} + \epsilon(h) \tag{3.24}$$

where $h > 0$ is some positive constant and $\epsilon(h) \in O(h)$ is an approximation error.

However, the important question here is the choice of the *"right"* value h. Although one can follow Equation (3.24) and choose h as small as possible, that approach leads to round-off error on a computer with floating-point arithmetic. There is a trade-off between round-off error and approximation error $\epsilon(h)$.

Automatic differentiation is an exact method, meaning that the derivative value can be obtained with an accuracy comparable to the symbolic inference of the derivatives followed by the calculating their numerical values. The fundamental idea behind automatic differentiation is the decomposition of the functions by the chain rule:

$$(f \circ g)' = f'(g)g'. \tag{3.25}$$

There are two modes of automatic differentiation: forward mode and reverse mode. The forward mode can be explained by introducing an algebra of *dual numbers*. A dual number is a number that can be expressed in the form $x + ys$, where $x, y \in \mathbb{R}$ are real and dual components, correspondingly, and s is a nilpotent element (i.e. an element that satisfies the property $s^2 = 0$).

The operations of addition and multiplication of dual numbers are defined as follows:

$$\begin{aligned}(x_1 + y_1 s) + (x_2 + y_2 s) &= (x_1 + x_2) + (y_1 + y_2)s \\ (x_1 + y_1 s)(x_2 + y_2 s) &= (x_1 x_2) + (x_1 y_2 + x_2 y_1)s + y_1 y_2 s^2 \\ &= (x_1 x_2) + (x_1 y_2 + x_2 y_1)s.\end{aligned} \tag{3.26}$$

Furthermore, any smooth function $f : \mathbb{R} \to \mathbb{R}$ can be extended to the dual numbers by introducing its Taylor series:

$$\begin{aligned}\bar{f}(x + ys) &= f(x) + f'(x)ys + \frac{f''(x)y^2 s^2}{2!} + \frac{f'''(x)y^3 s^3}{3!} + \dots \\ &= f(x) + f'(x)ys,\end{aligned} \tag{3.27}$$

where $\bar{f}$ denotes the extended to the dual numbers version of the function f.

The composition of functions $f \circ g$ is extended to the dual numbers in the following way:

$$\begin{aligned}(\bar{f} \circ \bar{g})(x + ys) = \bar{f}(\bar{g}(x + ys)) = \bar{f}(g(x) + g'(x)ys) = f(g(x)) + f'(g(x))g'(x)ys \\ = (f \circ g)(x) + (f \circ g)'(x)ys\end{aligned} \tag{3.28}$$

Note that

$$f(x)|_{x=a} + s\, f'(x)|_{x=a} = \bar{f}(a + s). \tag{3.29}$$

Hence, Equation (3.29) gives us a numerical method for evaluating derivatives values of any smooth function or composition of such functions. First, we extend these function f to the dual numbers. Then, we calculate the value of the extended function $\bar{f}(a+s)$. The result of such calculation is represented by a dual number. Finally, we obtain the function value $f(x)|_{x=a}$ and the derivative value $f'(x)|_{x=a}$ in real and dual components of the dual number, correspondingly.

3.4 Parallelization Strategy

Recall that the most of the computational resources are used by two algorithms:

1. Evaluation of the Jacobian matrix $J(x)$;

2. LSQR for solving linear least squares subproblem (3.18).

The computational time of these algorithms can be large if their implementation does not consider the Jacobian sparsity and parallel computation. However, a naive implementation using some parallel execution techniques and sparse linear algebra libraries is unlikely to succeed.

One must also take into account the access pattern to the data, with idea of reducing the cache miss rate. Choice of the target computing platform is also crucial.

Note that both of these algorithms can efficiently exploit so-called data parallelism: performing same operations on different pieces of source data in parallel. There are various programming models that implements data parallelism (e.g. CUDA[2], OpenCL[3], C++AMP[4], OpenACC[5]). Most of them are oriented towards GPU-accelerated computing.

3.4.1 GPU and CUDA

Modern GPUs are not only visual data processing units now. Due to their multiprocessor architecture they are capable to run many parallel algorithms more efficiently than conventional CPUs.

According to the NVIDIA website[6], a simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.

NVIDIA Compute Unified Device Architecture (CUDA) is a parallel computing platform, software framework and compiler infrastructure that provides C and C++ languages extensions, thereby making it easier to develop applications exploiting fine-grained and coarse-grained data and task parallelism.

CUDA implements a wide application program interface that provides such functionality as memory allocation, transferring and freeing, parallel execution, debugging, includes optimized versions of mathematical functions (e.g. the Bessel functions or the normal cumulative distribution function). There are various software products based on CUDA that provide highly-optimized algorithms such as the fast Fourier transform[7], basic linear algebra subroutines[8], random number generators[9] and high-level libraries of parallel algorithms and data structures[10].

[2] http://www.nvidia.ca/object/cuda_home_new.html
[3] https://www.khronos.org/opencl
[4] https://msdn.microsoft.com/en-us/library/hh265137.aspx
[5] http://www.openacc.org/
[6] http://www.nvidia.ca/object/what-is-gpu-computing.html
[7] https://developer.nvidia.com/cufft
[8] https://developer.nvidia.com/cublas
[9] https://developer.nvidia.com/curand
[10] https://developer.nvidia.com/thrust

```
beta = norm(u);
if beta > 0
  u     = (1/beta)*u;



  v = A'*u    - beta*v;
  alpha  = norm(v);

  if alpha > 0, v = (1/alpha)*v; end
end
```

(a) MATLAB

```
beta = cusp::blas::nrm2(u);
if (beta > 0) {
  cusp::blas::scal(u,1/beta);
  cusp::transpose(A,At);
  cusp::multiply(At,u,tmp);
  cusp::blas::axpy(v,tmp,-beta);
  std::swap(v,tmp);
  alpha = cusp::blas::nrm2(v);
  if (alpha > 0)
    cusp::blas::scal(v,1/alpha);
}
```

(b) C++

Figure 3.3: Line correspondence between LSQR codes written in MATLAB and C++.

3.4.2 Implementation Details

An inexact Levenberg-Marguardt and LSQR algorithms require effective implementation of linear algebra operations such as vector operation (additions, multiplication by scalar, ℓ_2-norm function), sparse matrix-vector multiplication and sparse matrix transposition. We use publicly available high-level C++ library CUSP[11] that implements these and many other functions and allows to exploit CUDA functionality without using low-level memory manipulations. CUSP also supports different sparse matrix formats including Compressed Row Storage.

Our CUDA implementation of LSQR is based on the Matlab version available on the Systems Optimization Laboratory website [43]. Figure 3.3 illustrates the line correspondence between code written in Matlab and C++ with CUSP. An inexact Levenberg-Marquardt algorithm is independently re-implemented based on the published materials [37].

CUDA extends the C++ language and allows to define functions (called *kernels*) that are intented to run on CUDA devices. Each kernel is assigned to be run by a logical computation unit (called *thread*). Threads are be grouped in *blocks*. By defining the dimensions of the block and the configuration of blocks (called a *grid*) to run programmer controls a fine-grained and coarse-grained parallelism. The process of mapping logical computation units to physical computational units are handled by the CUDA infrastructure. We implemented the Jacobian evaluation in a way, where each row of the Jacobian in Equation (3.22) is computed by one CUDA thread. Alternatively, one can implement the evaluation in a way, where each element of the Jacobian is mapped to one CUDA thread. We tested both and found former approach was faster than latter one. Figure 3.4a illustrates the chosen Jacobian evaluation strategy. Implementation of automatic differentiation is inspired by the Google Ceres Solver[12]. We exploit

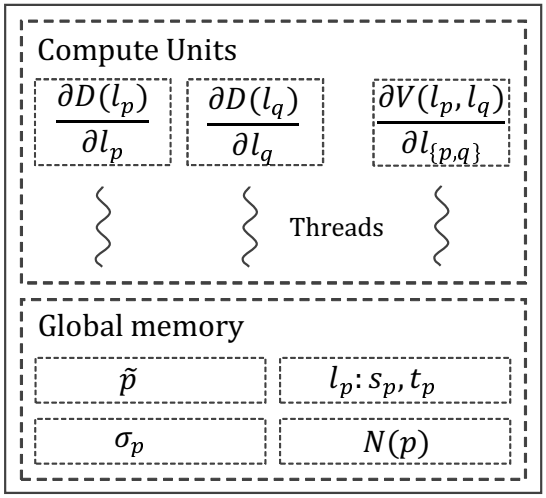
[11] http://cusplibrary.github.io/
[12] http://ceres-solver.org

the C++ template functions mechanism and operator overloading to implement an algebra of dual numbers. Then, we use it to numerically compute each component of the Jacobian without manual error-prone inference of partial derivatives.

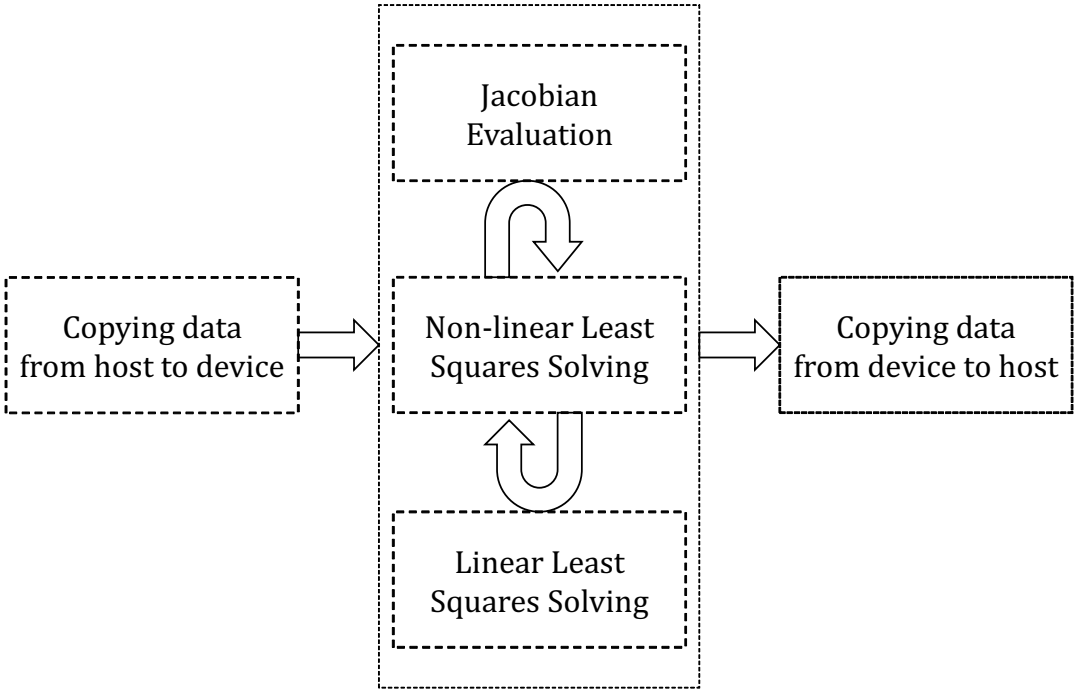
The most common bottle neck of the applications using CUDA is the transfer of data between device and host. We minimize this operations and requires to copy data from host to device and copy data back from device to host only once (see Figure 3.4b).

3.4.3 Experimental Result

We compare performance of our GPU-accelerated algorithm with performance of C++ prototype running on CPU. Even for quite old GPU device (NVIDIA Tesla M2070, 6 GB GDDR5) the total running time for our volume having size of $585 \times 525 \times 892$ voxels decreases from 2 weeks to half an hour. It is technically possible to achieve even better results by manual code optimization targeting a specific computation device.



(a) Jacobian Evaluation



(b) Algorithm Flow

Figure 3.4: GPU implementation of the curvature regularization framework. (a) Each row of the Jacobian matrix is evaluated by a separate thread. Threads interact with data by reading and writing from/to the global device memory. (b) illustrates the main steps of the framework.

Chapter 4

Validation

The purpose of validation is to compare the result obtained by our method with an accurate and precise measurements so-called *gold-standard*. Localization errors for detected bifurcations is one straightforward comparison criterion. Modern biomedical applications (see Subsection 1.1.2) also require accurate vascular tree topology validation. This chapter proposes validation methodologies using synthetic volumes. Our methodologies developed for validation with respect to ground truth can also be used for parameters tuning.

First, we review a standard method for synthetic generation of 3D vascular data based on vascular flow simulation (see Section 4.1). Then, we describe our validation procedure that consists of two steps: we match a ground truth tree to a testing tree and then compute various error statistics. Our matching algorithm is described in Section 4.2. The matching algorithm uses Fréchet distance [44]. Computing Fréchet distance [45] between corresponding branches of trees and the effective algorithm for its approximation [46] are discussed in Section 4.3. Our specific error statistics and their experimental evaluation for the results of the vessel centreline detection algorithm in Chapters 2 and 3 are reported in Section 4.4.

4.1 Ground Truth

This section outlines an algorithm for s generating vascular trees and synthetic 3D volumes proposed in [33, 34]. The generation of s vascular tree is based on simulating the vascular flow using realistic physics model and user-defined oxygen demand map. The algorithm produces tree models by randomly generating point locations and radii values and rejecting the candidates not satisfying these constraints. The generated tree data structure contains information about bifurcations locations, point connectivity and radii of vessels. Then, this ground truth tree is used to create 3D volumetric data by applying a special rasterization procedure. There are various types of noise that can also be added to the final result. Figure 4.1 illustrates two

synthetic vasculature volumes rendered along with ground truth trees.

4.2 Matching Problem

This section discusses the matching problem for two vascular trees: the *gold-standard tree* and the *testing tree* obtained by an algorithm. Let $T_1 = (V_1, E_1)$ denotes the gold-standard tree and $T_2 = (V_2, E_2)$ denotes the testing tree, where $V_n \subset \mathbb{R}^3$ are the corresponding node sets and $E_n = \{(p, q) \mid p, q \in V_n\}$ are the corresponding edge sets ($n = 1, 2$). To solve the matching problem, we look for a matching function

$$f(p) : V_1 \to V_2 \tag{4.1}$$

that relates each node $p \in V_1$ of the gold-standard tree to some node $f(p) \in V_2$ of the testing tree. Figure 4.2a shows an example of such a matching. However, none of the centerline extaction algorithms are ideal and some parts of the vascular tree can be unrecognized (see Figure 4.2b). Hence we relax the matching function $f(p)$ in the way where each node p can also be associated with a special value Λ, meaning *"no-matching"*:

$$f(p) : V_1 \to V_2 \cup \{\Lambda\}. \tag{4.2}$$

One straightforward way to define *best matching* between the gold-standard tree T_1 and the testing tree T_2 could be an optimal solution for a minimization of an *objective function*, for example:

$$\min_f \sum_{p \in V_1} D(p, f(p)), \tag{4.3}$$

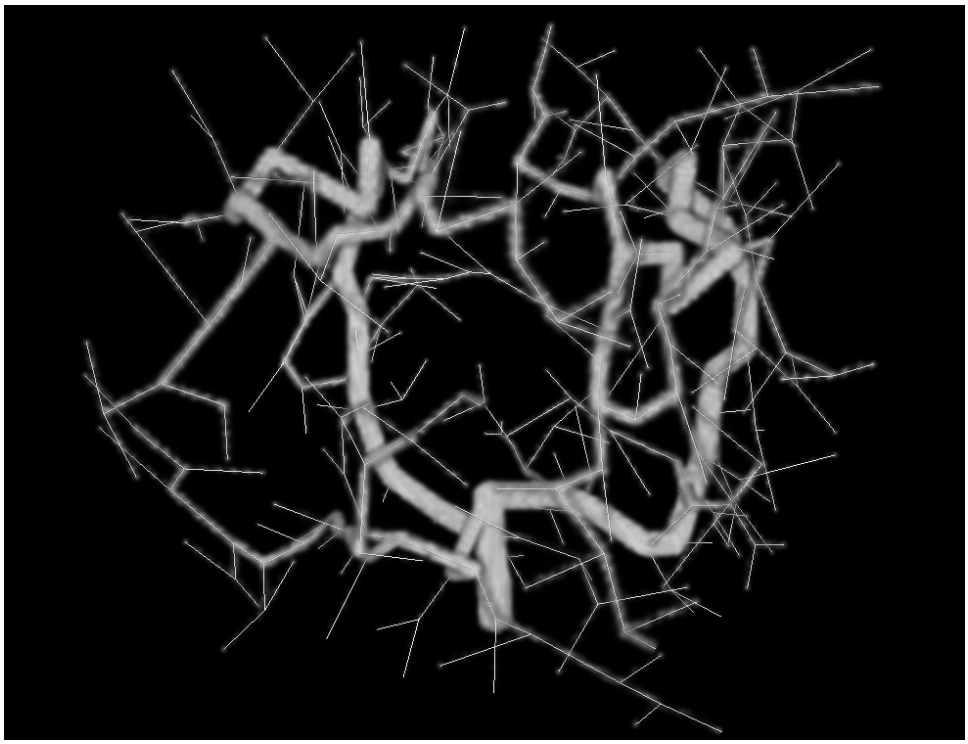
where

$$D(p, f(p)) = \begin{cases} \|p - f(p)\| & \text{if } f(p) \in V_2 \\ \alpha & \text{if } f(p) = \Lambda \end{cases} \tag{4.4}$$

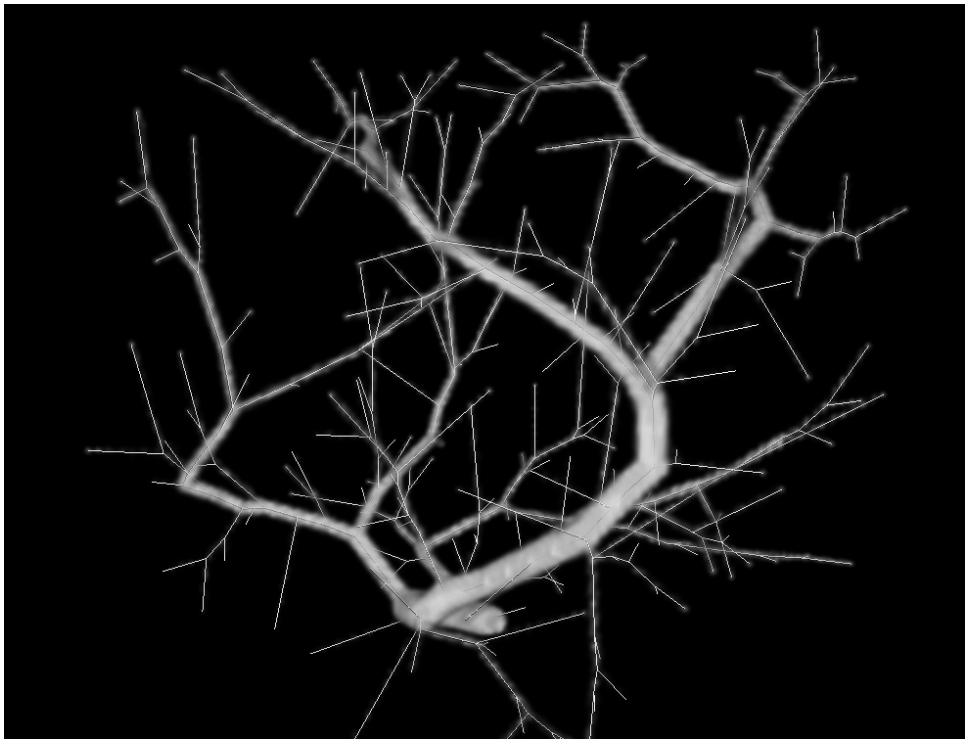
denotes *cost* of assigning value $f(p) \in V_2 \cup \{\Lambda\}$ to node $p \in V_1$. Note that $\|p - f(p)\|$ is the Euclidean distance between p and the corresponding point $f(p)$. Constant penalty $\alpha > 0$ defines a maximum allowed error (see Figure 4.2c). Equation (4.3) can be equivalently represented by

$$\min_f \sum_{p \in V_1 : f(p) \in V_2} \|p - f(p)\| + \alpha \sum_{p \in V_1 : f(p) = \Lambda} 1, \tag{4.5}$$

where the first sum measures the total *geometric error* between matched nodes from two trees T_1 and T_2 while the second sum measures the *cardinality* of the unmatched part and α defines



(a)



(b)

Figure 4.1: Two examples of generated synthetic vascular volumes rendered along with ground truth trees.

a trade-off between them. Similar to Gorelick *et al.* [47], the unconstrained optimization Equation (4.3) for some $\alpha > 0$ can be represented as a Lagrangian for the following constrained optimization problem, for some $\epsilon > 0$:

$$\max_{f:GE(f)<\epsilon} C(f), \tag{4.6}$$

where

$$GE(f) = \sum_{p \in V_1 : f(p) \in V_2} \|p - f(p)\| \tag{4.7}$$

denotes the total *geometric error* between matched pairs of nodes in V_1 and V_2 and

$$C(f) = |\{p \in V_1 : f(p) \in V_2\}| \tag{4.8}$$

denotes the *cardinality* of the matched part.

Note that the geometric error function in Equation (4.7) does not take into account the connectivity of points in the ground truth and the testing trees (see Figure 4.2d). Due to specific applications of our centerline estimation method (see Subsection 1.1.2) the vascular tree topology also has to be validated.

4.3 Fréchet Distance Measure

This section describes the Fréchet distance between two polygonal curves [45]. We discussed earlier that the geometric error based on the trees displacement only is not sufficient for our applications. Hence we introduce geometric error function that incorporates positions errors and connectivity errors. This geometric error function is based on the Fréchet distance that was introduces by Fréchet [44] as a measure of similarity between two curves in a metric space.

First, we introduce geometric error function for matching $f : V_1 \to V_2$ that takes into account connectivity of points in both trees:

$$GE(f) = \sum_{\substack{(p,q) \in E_1 : f(p), f(q) \in V_2 \\ f(p) \text{ and } f(q) \text{ are connected}}} V(p, q, f(p), f(q)) \tag{4.9}$$

The objective of the function $V(p, q, f(p), f(q))$ is to measure the distance between two curves: the line starting at point p and ending at point q and the piecewise linear curve including all points on the path from point $f(p)$ to point $f(q)$. This problem often arises also in a context of object and shape recognition. The are two well-known such distance measures:

- **Hausdorff distance**: For two polygonal chains A and B defined by the corresponding

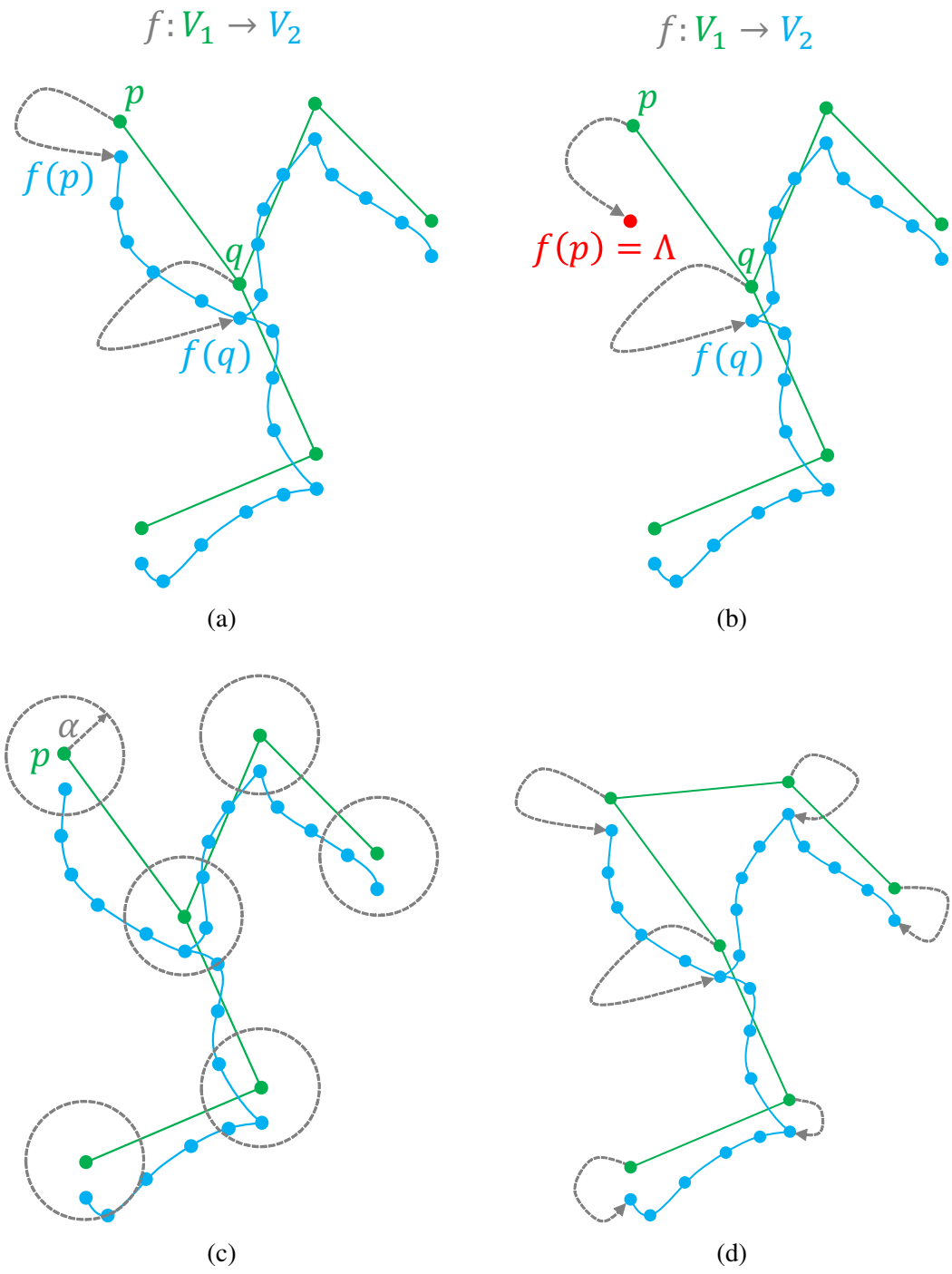


Figure 4.2: (a,b) illustrates an example of assigning labels $f(p)$ and $f(q)$ to nodes p and q, correspondingly. Note that path between nodes $f(p)$ and $f(q)$ exists in (a). (c) shows maximum allowed error α for the matching problem given in Equation (4.3). (d) demonstrates the solution to the matching problem given in Equation (4.3) that does not consider tree topology.

point sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ (as it is illustrated in Figure 4.3), the Hausdorff distance is defined as a maximum distance between each curve and the nearest point of another curve. Thus, Hausdorff distance can be represented by

$$\max(D_{A\to B}, D_{B\to A}) \tag{4.10}$$

where

$$\begin{aligned} D_{A\to B} &= \max_{1\le i\le n} \min_{1\le j\le m} \|a_i - b_j\| \\ D_{B\to A} &= \max_{1\le j\le m} \min_{1\le i\le n} \|a_i - b_j\| \end{aligned} \tag{4.11}$$

are the *directional Hausdorff distances* between curves A and B. Note that although Hausdorff distance takes into account positions of all points defining polygonal chains, it does not use the knowledge about the order in which their points appear in both curves.

- **Fréchet distance**: According to Eiter *et al.* [46] the intuitive definition of the Fréchet distance would be to picture the situation when a man is walking a dog. Their movements constraints are as follows: the man can move on the first curve and the dog can move on the second curve. In that case, the *weak Fréchet distance* is defined as a minimum of leash length that allows both man and dog walking from one endpoints of the curves to another. The *strong Fréchet distance* is similar, but backtracking is not allowed.

  Let two curves A and B in a metric space are defined as a continuous functions

$$\begin{aligned} A(s) &: [0, l_A] \to \mathbb{R}^3 \\ B(t) &: [0, l_B] \to \mathbb{R}^3 \end{aligned} \tag{4.12}$$

  where l_A, l_B are lengths of curves A and B, respectively. Then, the *strong Fréchet distance* is given as:

$$\inf_{\alpha,\beta} \max_{s\in[0,1]} \|A(\alpha(s)) - B(\beta(s))\|, \tag{4.13}$$

  where $\alpha : [0,1] \to [0, l_A]$ and $\beta : [0,1] \to [0, l_B]$ are two continuous non-decreasing functions. Fréchet distance measures are based on two factors: the location of points and the position along the curves. Alt *et al.* [45] proposed an algorithm for evaluating Fréchet distance between polygonal chains. However, the algorithm is quite complicated. Our implementation uses the efficient algorithm given by Eiter *et al.* [46] for computing an approximation of Fréchet distance, also known as the *discrete Fréchet distance*.

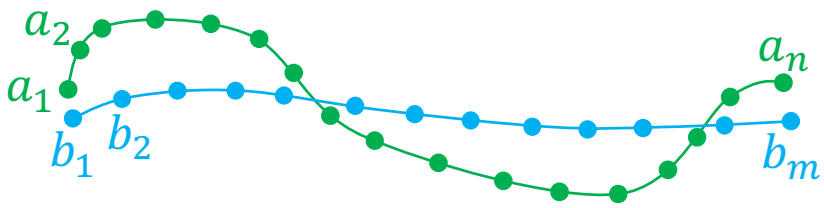


Figure 4.3: Example of two polygonal chains defined by corresponding point sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$.

4.4 Experimental Evaluation on Synthetic Data

This section describes our error statistics for the validation of centerline extraction algorithms. We define the *best* matching between the ground truth tree and a testing tree as an optimal solution of the following constrained optimization problem:

$$\max_{f:GE(f)<\epsilon} L(f) \tag{4.14}$$

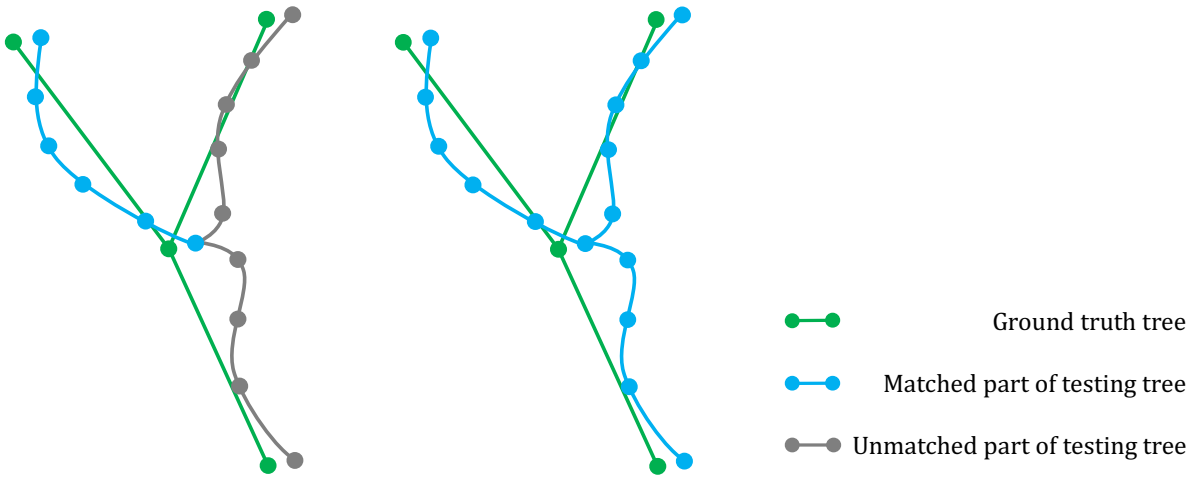
where the geometric error $GE(f)$ is defined by Equation (4.9) and is based on the Fréchet distance measure (see Section 4.3) and

$$L(f) = \sum_{\substack{(p,q)\in E_1 : f(p), f(q)\in V_2 \\ f(p) \text{ and } f(q) \text{ are connected}}} \|p - q\|, \tag{4.15}$$

which defines the total length of the matched part of the ground truth tree. Thus, each value $\epsilon > 0$ defines the maximum allowed geometric error for matching and the best matching f maximizes the total length of matched part (see Figure 4.4). We additionally require the following constraint: each node p is allowed to be assigned either to one of k nearest neighbors of point p or to a "no-matching" value Λ. Indeed, there is a little point to look for a corresponding node $f(p)$ that has locations too far from p. The optimal solution of the optimization problem given in Equation (4.14) can be found by applying an efficient version of Viterbi algorithm [48]. We refer the interested reader to appendix A for a detailed explanation. The value $k = 17$ is found adequate for all our experiments.

We use the VascuSynth software [33, 34] for generating 500 synthetic vascular trees and 3D volumes (e.g., see Figures 4.1a and 4.1b). Then, we compare the tree data structures obtained by the MATLAB implementation of Kroon[1] of an accurate fast marching algorithm for skeletons computation [8, 9, 10] (see Figure 4.5), our centerline estimation methods without (see Figure 4.6) and with (see Figure 4.7) curvature-based regularization.

[1] `http://www.mathworks.com/matlabcentral/fileexchange/24531-accurate-fast-marching`



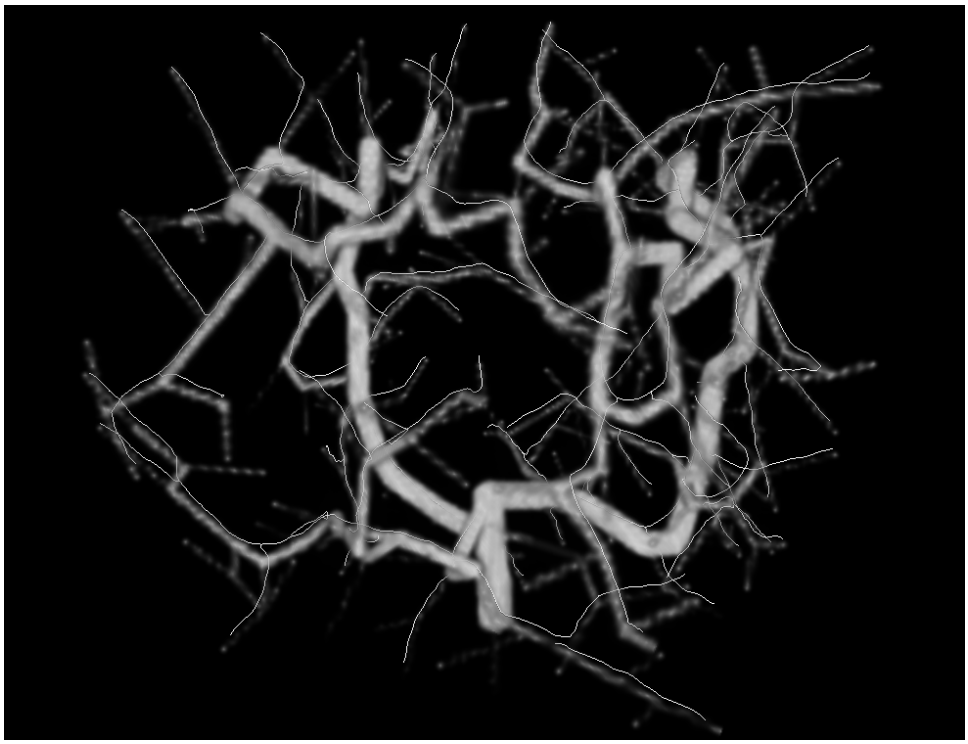
(a) (b)

Figure 4.4: (a) We believe that the smaller geometric error GE allowed the smaller parts of trees can be matched. (b) On other hand, the larger geometric error GE allows more freedom in matching and consequently the larger parts of the trees can be matched.

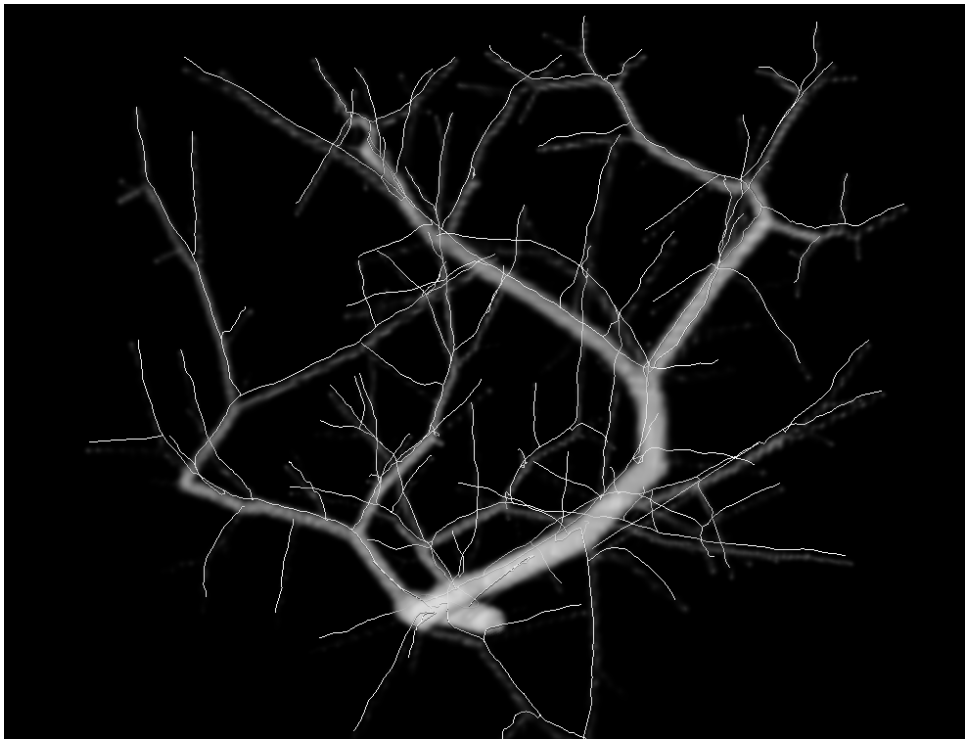
Figure 4.8 demonstrates the total matched length $L(f)$ plotted against the geometric error $GE(f)$. Our estimation method for any value of the geometric error GE gives greater length L of matched part than skeleton method. Curvature regularization shows some improvement if compared with our method without such regularization. In the future, we plan to improve tree connectivity by using vessel extrapolation and expect even better results. Our curvature regularization not only removes noise, but also gives good orientations that we can use for vessel extrapolation.

4.5 Experimental Evaluation on Real Data

We tune parameters of our algorithm using the validation results in Section 4.4. Then, we run the algorithm for the CT cardiac volume (see Figure 4.9a). The size of the volume is $585 \times 525 \times 892$ voxels. We reconstruct the positions of the centerpoints of the vessels as well as an information about connectivity of these points (see Figure 4.9b). The tree structure allows to measure angles between any two adjacent vessels, their diameter or their length. Our algorithm is capable to reconstruct very thin vessels at near-capillary level (see Figures 4.9c and 4.9d).

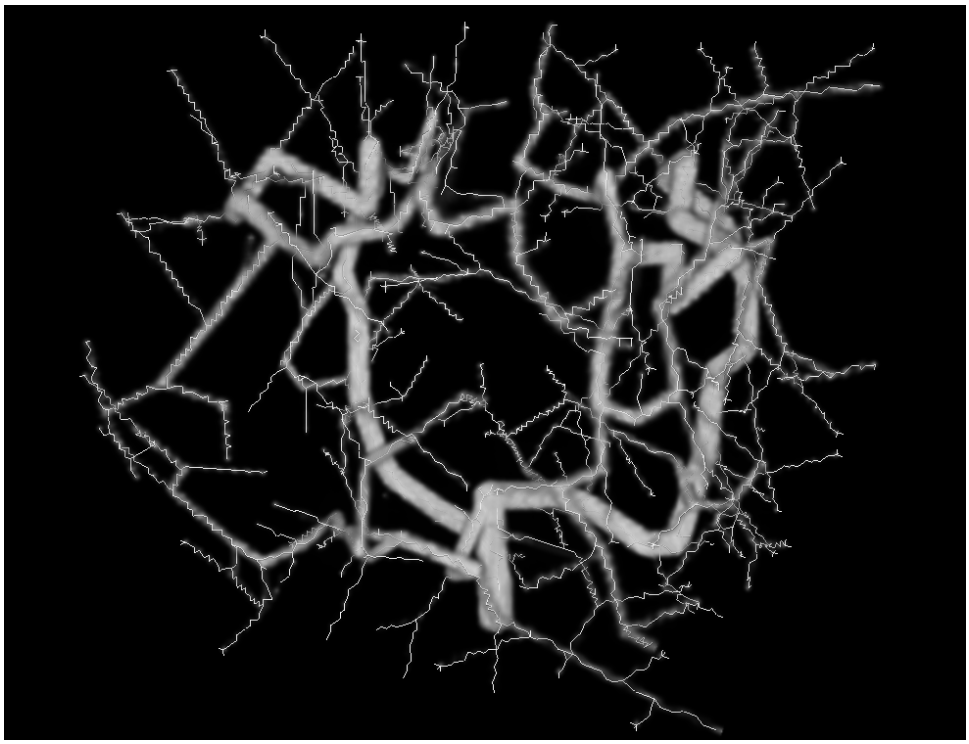


(a)



(b)

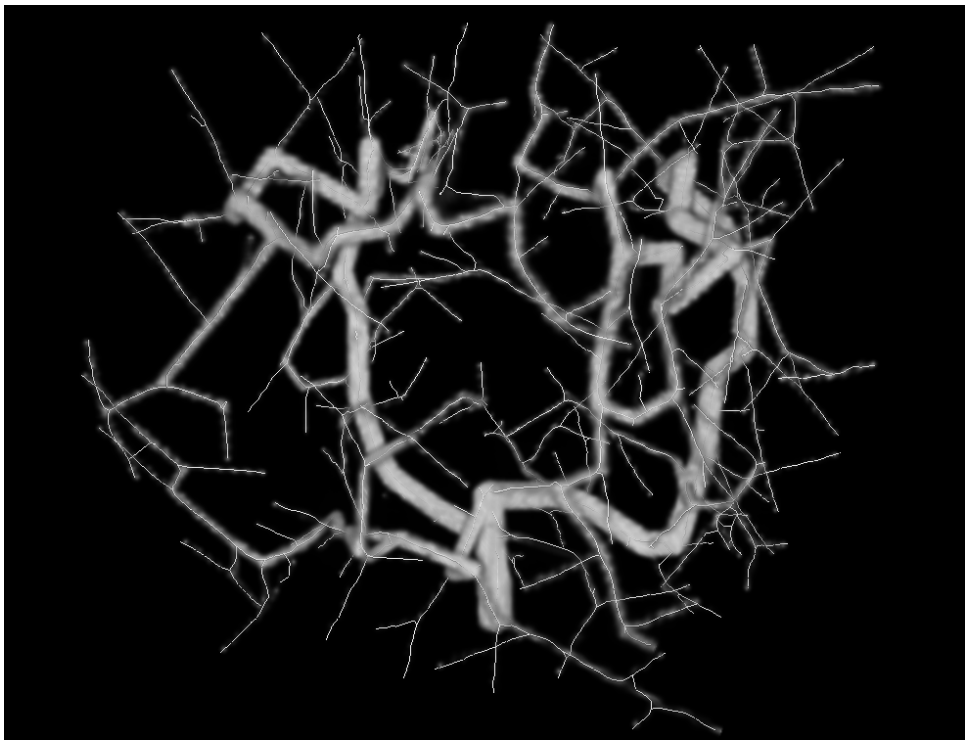
Figure 4.5: Results obtained by the accurate fast marching algorithm implementation of Kroon for skeleton computation. (a) Dataset 1 (see Figure 4.1a) (b) Dataset 2 (see Figure 4.1b)



(a)

(b)

Figure 4.6: Results obtained by centerline estimation algorithm without curvature regularization [12]. (a) Dataset 1 (see Figure 4.1a) (b) Dataset 2 (see Figure 4.1b)



(a)

(b)

Figure 4.7: Results obtained by our centerline estimation algorithm with curvature-based regularization. (a) Dataset 1 (see Figure 4.1a) (b) Dataset 2 (see Figure 4.1b)

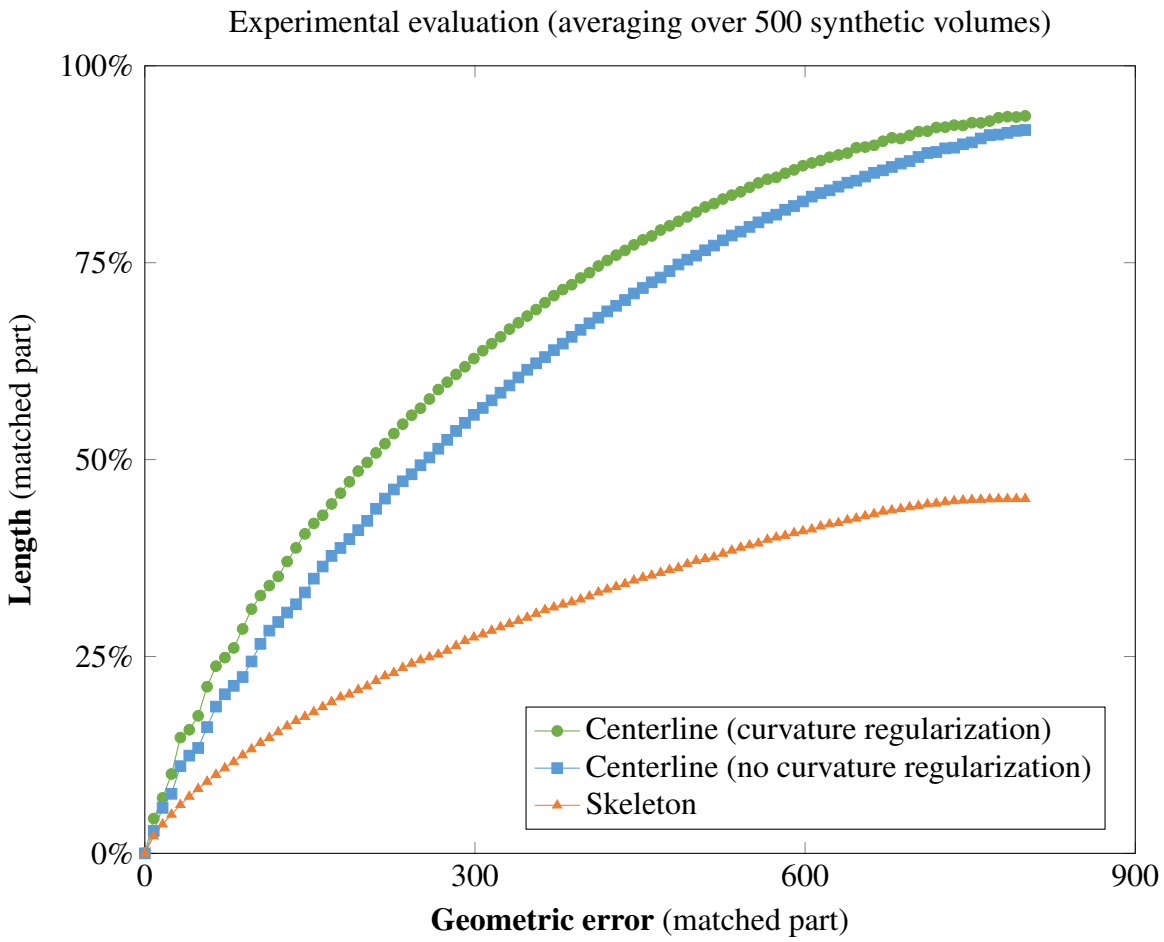
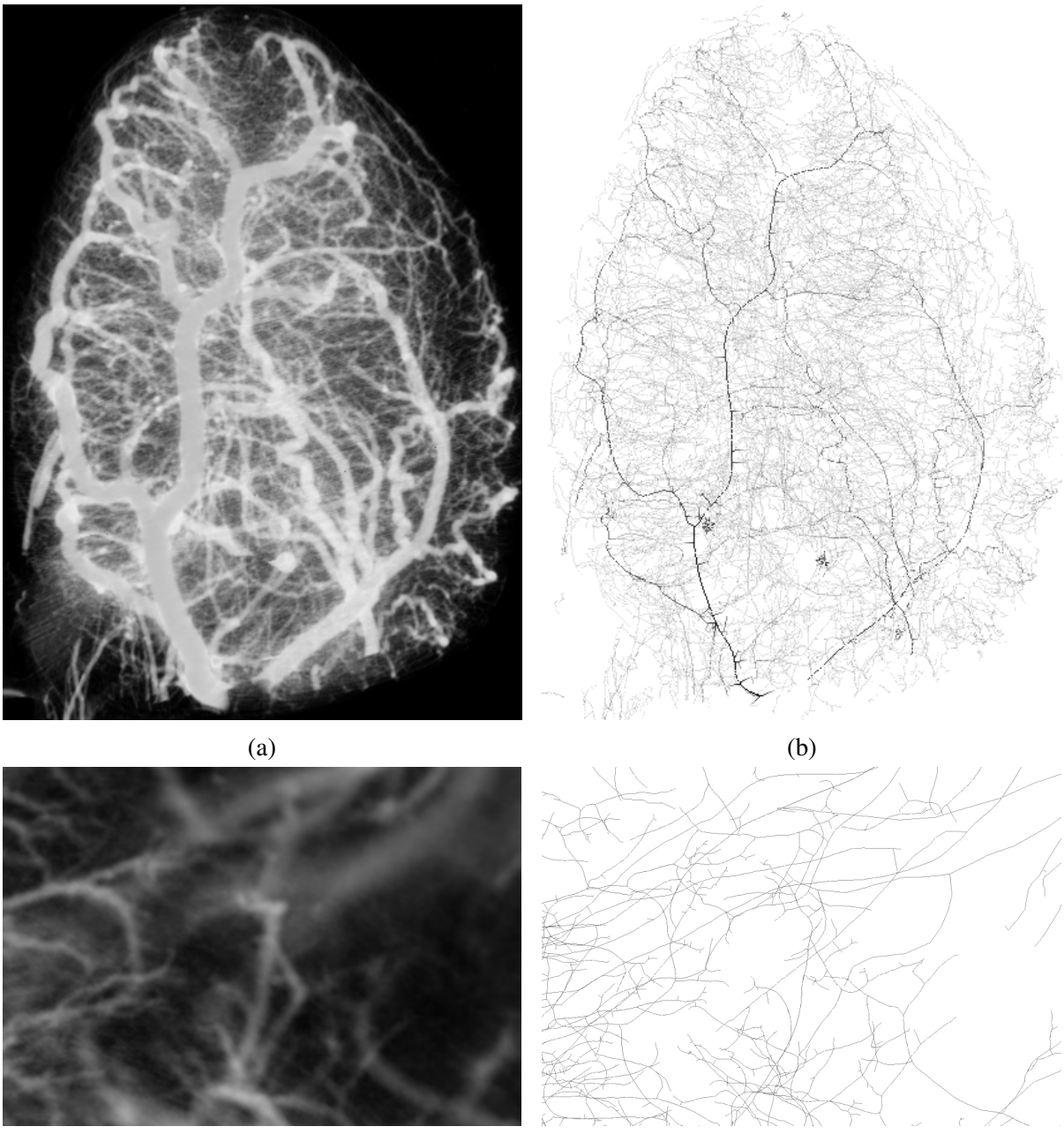


Figure 4.8: Comparison results obtained by three centerline estimation methods: our centerline estimation method with curvature-based regularization, centerline estimation method based on ideas of Canny-edge detector [12], and accurate fast marching algorithm implementation of Kroon for skeleton-based estimation.



(a) (b)

(c) (d)

Figure 4.9: Results obtained by our centerline estimation algorithm for real CT data. (a) and (c) show volume rendering of the raw data volume at the coarse and the fine scale while (b) and (d) shows the corresponding visualizations of the output of our algorithm.

Chapter 5

Conclusion and Future Work

This thesis proposed methodologies for solving the challenging problem of vessel tree structure centreline reconstruction from ultra high resolution micro-CT volumes. Our method avoids explicit segmentation of vessel boundaries. In contrast, we directly extract the vascular tree by regularizing the curvature of vessel centerline. We developed a GPU-based implementation of the Levenberg-Marquardt algorithm and applied it to our curvature regularization problem. We also proposed a validation technique using synthetic vessel images. Results of comparison our approach with standard medial axis methods are promising.

In the future we plan to use our matching mechanism inside validation procedure for parameters tuning of our curvature-based tree extraction method. Also, we plan to improve tree connectivity by using vessel extrapolation using smoothed tangent directions.

Bibliography

[1] J. Hsieh, "Computed tomography: principles, design, artifacts, and recent advances," SPIE Bellingham, WA, 2009.

[2] J. Sijbers and A. Postnov, "Reduction of ring artefacts in high resolution micro-ct reconstructions," *Physics in Medicine and Biology*, vol. 49, no. 14, p. N247, 2004.

[3] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever, "Multiscale vessel enhancement filtering," in *Medical Image Computing and Computer-Assisted Interventation MICCAI 98*, pp. 130–137, Springer, 1998.

[4] Y. Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in nd images," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1, pp. 105–112, IEEE, 2001.

[5] S. Osher and R. Fedkiw, *Level set methods and dynamic implicit surfaces*, vol. 153. Springer Science & Business Media, 2006.

[6] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[7] S. Heber, R. Ranftl, and T. Pock, "Approximate envelope minimization for curvature regularity," in *Computer Vision–ECCV 2012. Workshops and Demonstrations*, pp. 283–292, Springer, 2012.

[8] J. A. Bærentzen, "On the implementation of fast marching methods for 3d lattices," tech. rep., 2001.

[9] S. M. Hassouna, A. Farag, *et al.*, "Multistencils fast marching methods: A highly accurate solution to the eikonal equation on cartesian domains," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 9, pp. 1563–1574, 2007.

[10] R. Van Uitert and I. Bitter, "Subvoxel precise skeletons of volumetric data based on fast marching methods," *Medical physics*, vol. 34, no. 2, pp. 627–638, 2007.

[11] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[12] Y. Zhong, "Extracting vessel structure from 3d image data," 2014.

[13] T. L. Kline, M. Zamir, and E. L. Ritman, "Accuracy of microvascular measurements obtained from micro-ct images," *Annals of biomedical engineering*, vol. 38, no. 9, pp. 2851–2864, 2010.

[14] T. L. Kline, M. Zamir, and E. L. Ritman, "Relating function to branching geometry: a micro-ct study of the hepatic artery, portal vein, and biliary tree," *Cells Tissues Organs*, vol. 194, no. 5, pp. 431–442, 2011.

[15] V. Kolmogorov, Y. Boykov, and C. Rother, "Applications of parametric maxflow in computer vision," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[16] P. Strandmark and F. Kahl, "Curvature regularization for curves and surfaces in a global optimization framework," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pp. 205–218, Springer, 2011.

[17] K. Bredies, T. Pock, and B. Wirth, "Convex relaxation of a class of vertex penalizing functionals," *Journal of mathematical imaging and vision*, vol. 47, no. 3, pp. 278–302, 2013.

[18] C. Nieuwenhuis, E. Toeppe, L. Gorelick, O. Veksler, and Y. Boykov, "Efficient squared curvature," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 4098–4105, IEEE, 2014.

[19] T. Schoenemann, F. Kahl, S. Masnou, and D. Cremers, "A linear framework for region-based image segmentation and inpainting involving curvature penalization," *International Journal of Computer Vision*, vol. 99, no. 1, pp. 53–68, 2012.

[20] D. Marin, Y. Boykov, and Y. Zhong, "Thin structure estimation with curvature regularization," *arXiv preprint arXiv:1506.04654*, 2015.

[21] K. Siddiqi and S. Pizer, *Medial representations: mathematics, algorithms and applications*, vol. 37. Springer Science & Business Media, 2008.

[22] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis, "Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images," *Medical image analysis*, vol. 2, no. 2, pp. 143–168, 1998.

[23] Y. Kitamura, Y. Li, W. Ito, and H. Ishikawa, "Data-dependent higher-order clique selection for artery–vein segmentation by energy minimization," *International Journal of Computer Vision*, pp. 1–17, 2015.

[24] J. Yi and J. B. Ra, "A locally adaptive region growing algorithm for vascular segmentation," *International Journal of Imaging Systems and Technology*, vol. 13, no. 4, pp. 208–214, 2003.

[25] B. E. Chapman, J. O. Stapelton, and D. L. Parker, "Intracranial vessel segmentation from time-of-flight mra using pre-processing of the mip z-buffer: accuracy of the zbs algorithm," *Medical Image Analysis*, vol. 8, no. 2, pp. 113–126, 2004.

[26] H. Mirzaalian and G. Hamarneh, "Vessel scale-selection using mrf optimization," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 3273–3279, IEEE, 2010.

[27] J. I. Orlando and M. Blaschko, "Learning fully-connected crfs for blood vessel segmentation in retinal images," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2014*, pp. 634–641, Springer, 2014.

[28] J. Stühmer and D. Cremers, "A fast projection method for connectivity constraints in image segmentation," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pp. 183–196, Springer, 2015.

[29] N. D. Forkert, A. Schmidt-Richberg, J. Fiehler, T. Illies, D. Möller, D. Säring, H. Handels, and J. Ehrhardt, "3d cerebrovascular segmentation combining fuzzy vessel enhancement and level-sets with anisotropic energy weights," *Magnetic resonance imaging*, vol. 31, no. 2, pp. 262–271, 2013.

[30] F. Benmansour and L. D. Cohen, "Tubular structure segmentation based on minimal path method and anisotropic enhancement," *International Journal of Computer Vision*, vol. 92, no. 2, pp. 192–210, 2011.

[31] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[32] C. Olsson and Y. Boykov, "Curvature-based regularization for surface approximation," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1576–1583, IEEE, 2012.

[33] G. Hamarneh and P. Jassi, "VascuSynth: Simulating Vascular Trees for Generating Volumetric Image data with Ground Truth Segmentation and Tree Analysis," *Computerized Medical Imaging and Graphics*, vol. 34, no. 8, pp. 605–616, 2010.

[34] P. Jassi and G. Hamarneh, "VascuSynth: Vascular Tree Synthesis Software," *Insight Journal*, vol. January-June, pp. 1–12, 2011.

[35] H. Blum, "A transformation for extracting descriptors of shape," 1967.

[36] C. Olsson, J. Ulén, and Y. Boykov, "In defense of 3d-label stereo," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 1730–1737, IEEE, 2013.

[37] S. Wright and J. N. Holt, "An inexact Levenberg-Marquardt method for large sparse nonlinear least squares," *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, vol. 26, no. 04, pp. 387–403, 1985.

[38] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Transactions on Mathematical Software (TOMS)*, vol. 8, no. 1, pp. 43–71, 1982.

[39] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008.

[40] K. Levenberg, "A method for the solution of certain non–linear problems in least squares," 1944.

[41] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[42] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[43] "Systems Optimization Laboratory." `http://web.stanford.edu/group/SOL/software/lsqr/`.

[44] M. M. Fréchet, "Sur quelques points du calcul fonctionnel," *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, vol. 22, no. 1, pp. 1–72, 1906.

[45] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[46] T. Eiter and H. Mannila, "Computing discrete Fréchet distance," tech. rep., Citeseer, 1994.

[47] L. Gorelick, F. R. Schmidt, and Y. Boykov, "Fast trust region for segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 1714–1721, IEEE, 2013.

[48] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260–269, 1967.

[49] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

Appendix A

Viterbi Algorithm

This appendix overviews an efficient algorithm for solving our discrete labeling problem as given in Equation (4.14). Although one might recognize that this is a variant of the well-known Viterbi algorithm used in relation to hidden Markov models, we illustrate the algorithm in the context of our matching problem, without touching on probabilistic models. We refer the reader interested in that particular topic to the original work done by Viterbi [48].

A.1 Viterbi Algorithm for Chains

This section illustrates the Viterbi algorithm for chains. We assume that ground truth tree in Equation (4.14) is a chain-like tree (see Figure A.1a). We do not make any assumption about T_2, so it can be any arbitrary tree (or forest).

Let $f_1, f_2 \dots f_n$ denote label variables corresponding to nodes $v_1, v_2 \dots v_n$ and $n = |V_1|$ be the number of nodes in tree T_1. Assume also without loss of generality that the nodes of T_1 are numbered in walking order, starting at v_1 and ending at v_n (e.g. as it is shown in Figure A.1a).

Consider the following labeling problem:

$$\min_{f_1, f_2 \dots f_n} \sum_{i=1}^{n} E_1(v_i, f_i) + \sum_{i=1}^{n-1} E_2(v_i, v_{i+1}, f_i, f_{i+1}), \tag{A.1}$$

where $E_1(v, f(v))$, $E_2(u, v, f(u), f(v))$ are some arbitrary unary and pairwise cost functions (not necessary defined as it has done in earlier section).

We solve labeling problem (A.1) in dynamic programming way by transforming the original problem into a sequence of simpler subproblems.

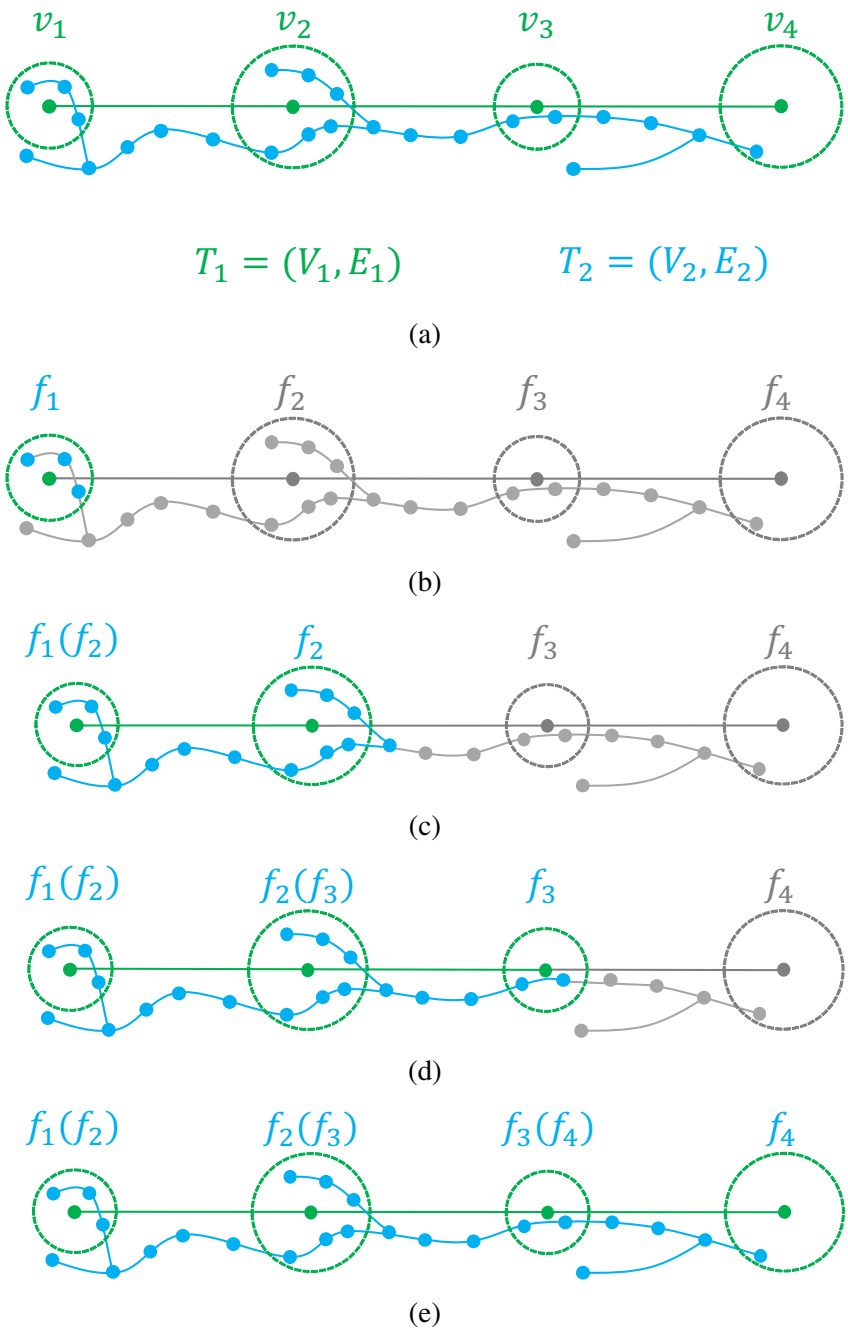


Figure A.1: Viterbi algorithm for a chain. (a) illustrates the matching problem for the ground truth tree T_1 and testing tree T_2. (b-e) subsequent steps of the forward pass calculation of the Viterbi algorithm.

- We introduce function H_1 that contains single term in Equation (A.1):

$$H_1(f_1) = E_1(v_1, f_1). \tag{A.2}$$

Note that H_1 depends *only* on label variable f_1. Figure A.1b illustrates elements of trees $T_1,\ T_2$ related to H_1.

- We introduce function H_2 as follows:

$$H_2(f_2) = E_1(v_2, f_2) + \min_{f_1} H_1(f_1) + E_2(v_1, v_2, f_1, f_2). \tag{A.3}$$

Note that H_2 depends *only* on label variable f_2, while f_1 is chosen as an optimal solution of optimization subproblem for each value of f_2:

$$f_1(f_2) = \arg\min_{f_1} H_1(f_1) + E_2(v_1, v_2, f_1, f_2). \tag{A.4}$$

Figure A.1c illustrates the elements of trees T_1 and T_2 which are parts of H_2.

- Continuing recursively (see Figure A.1d) we get:

$$H_{n-1}(f_{n-1}) = E_1(v_{n-1}, f_{n-1}) + \min_{f_{n-2}} H_{n-2}(f_{n-2}) + E_2(v_{n-2}, v_{n-1}, f_{n-2}, f_{n-1}), \tag{A.5}$$

where optimal choice f_{n-2} depends on f_{n-1}:

$$f_{n-2}(f_{n-1}) = \arg\min_{f_{n-2}} H_{n-2}(f_{n-2}) + E_2(v_{n-2}, v_{n-1}, f_{n-2}, f_{n-1}). \tag{A.6}$$

- Finally, we have function H_n that depends only on the label f_n (see Figure A.1e)

$$H_n(f_n) = E_1(v_n, f_n) + \min_{f_{n-1}} H_{n-1}(f_{n-1}) + E_2(v_{n-1}, v_n, f_{n-1}, f_n). \tag{A.7}$$

One can see that the minimum of function H_n equals to the minimum of the original problem (A.1), is:

$$\begin{aligned}
\min_{f_n} H_n(f_n) = \\
&= \min_{f_n} E_1(v_n, f_n) + \min_{f_{n-1}} E_2(v_{n-1}, v_n, f_{n-1}, f_n) + H_{n-1}(f_{n-1}) \\
&= \min_{f_n} E_1(v_n, f_n) + \min_{f_{n-1}} E_2(v_{n-1}, v_n, f_{n-1}, f_n) + E_1(v_{n-1}, f_{n-1}) + \min_{f_{n-2}} H_{n-2}(f_{n-2}) \\
&= \min_{f_n} E_1(v_n, f_n) + \min_{f_{n-1}} E_2(v_{n-1}, v_n, f_{n-1}, f_n) + E_1(v_{n-1}, f_{n-1}) + \min_{f_{n-2}} E_1(v_{n-2}, f_{n-2}) + \dots \\
&= \min_{f_1, f_2 \dots f_n} \sum_{i=1}^{n} E_1(v_i, f_i) + \sum_{i=1}^{n-1} E_2(v_i, v_{i+1}, f_i, f_{i+1}).
\end{aligned} \tag{A.8}$$

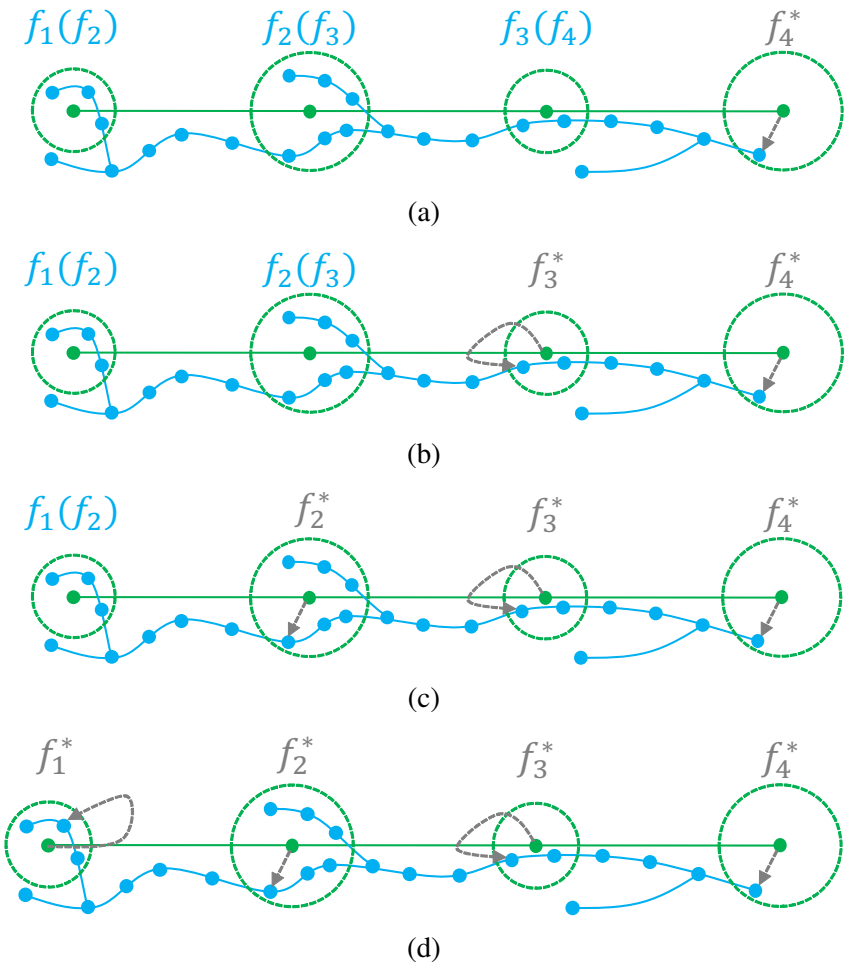


Figure A.2: The Viterbi algorithm for a chain. (a-d) subsequent steps of the backward pass of the Viterbi algorithm.

Our objective is to recursively solve the subproblems one by one.

- We start with the subproblem for H_n:

$$f_n^* = \arg\min_{f_n} H_n(f_n) \tag{A.9}$$

  Figure A.2a shows optimal choice f_n^*.

- Since f_{n-1} depends on f_n we can find optimal value f_n^*:

$$f_{n-1}^* = \arg\min_{f_{n-1}} H_{n-1}(f_{n-1}) + E_2(v_{n-1}, v_n, f_{n-1}, f_n^*) \tag{A.10}$$

  Figure A.2b shows optimal choices f_{n-1}^* and f_n^*.

- We continue recursively (see Figure A.2c) until we get:

$$f_2^* = \arg\min_{f_2} H_2(f_2) + E_2(v_2, v_3, f_2, f_3^*). \tag{A.11}$$

- Finally, we find optimal value of last label (see Figure A.2d)

$$f_1^* = \arg\min_{f_1} H_1(f_1) + E_2(v_1, v_2, f_1, f_2^*). \tag{A.12}$$

A.2 Viterbi Algorithm for Tree

This section describes the same optimization technique for solving our original matching problem given in Equation (4.14), where T_1 is an arbitrary tree. Felzenszwalb *et al.* [49] used the same optimization approach in the context of object recognition. Consider the following matching problem:

$$\min_{f_1, f_2 \ldots f_n} \sum_{i=1}^{n} E_1(v_i, f_i) + \sum_{\{v_i, v_j\} \in E_1} E_2(v_i, v_j, f_i, f_j), \tag{A.13}$$

where T_1 and T_2 denote ground truth and testing tree, respectively. We assume that the nodes of T_1 are numbered in level-order, where we visit every node on a level before going to a lower level (Figure A.3a).

We decompose the original problem into a sequence of smaller subproblems in a similar way.

- For each node v_i on the lowest level, we obtain the functions as (see Figure A.3b):

$$\begin{aligned} H_7(f_7) &= E_1(v_7, f_7), \\ H_8(f_8) &= E_1(v_8, f_8) \text{ and} \\ H_9(f_9) &= E_1(v_9, f_9). \end{aligned} \tag{A.14}$$

- For each node v_i on an upper level (see Figure A.3c):

$$\begin{aligned} H_4(f_4) &= E_1(v_4, f_4), \\ H_5(f_5) &= E_1(v_5, f_5) + \min_{f_7} H_7(f_7) + E_2(v_5, v_7, f_5, f_7) + \min_{f_8} H_8(f_8) + E_2(v_5, v_8, f_5, f_8), \\ H_6(f_6) &= E_1(v_6, f_6) + \min_{f_9} H_9(f_9) + E_2(v_6, v_9, f_6, f_9), \end{aligned} \tag{A.15}$$

and

$$\begin{aligned} f_7(f_5) &= \arg\min_{f_7} H_7(f_7) + E_2(v_5, v_7, f_5, f_7), \\ f_8(f_5) &= \arg\min_{f_8} H_8(f_8) + E_2(v_5, v_8, f_5, f_8), \\ f_9(f_6) &= \arg\min_{f_9} H_9(f_9) + E_2(v_6, v_9, f_6, f_9). \end{aligned} \tag{A.16}$$

- We continue on toward an upper level (see Figure A.3d)

$$\begin{aligned} H_2(f_2) &= E_1(v_2, f_2) + \min_{f_4} H_4(f_4) + E_2(v_2, v_4, f_2, f_4) + \min_{f_5} H_5(f_5) + E_2(v_2, v_5, f_2, f_5), \\ H_3(f_3) &= E_1(v_3, f_3) + \min_{f_6} H_6(f_6) + E_2(v_3, v_6, f_3, f_6), \end{aligned} \tag{A.17}$$

and

$$\begin{aligned} f_4(f_2) &= \arg\min_{f_4} H_4(f_4) + E_2(v_2, v_4, f_2, f_4), \\ f_5(f_2) &= \arg\min_{f_5} H_5(f_5) + E_2(v_2, v_5, f_2, f_5), \\ f_6(f_3) &= \arg\min_{f_6} H_6(f_6) + E_2(v_3, v_6, f_3, f_6). \end{aligned} \tag{A.18}$$

- Finally, at the highest level (see Figure A.3e) we get

$$H_1(f_1) = E_1(v_1, f_1) + \min_{f_2} H_2(f_2) + E_2(v_1, v_2, f_1, f_2) + \min_{f_3} H_3(f_3) + E_2(v_1, v_3, f_1, f_3) \tag{A.19}$$

and

$$\begin{aligned} f_2(f_1) &= \arg\min_{f_2} H_2(f_2) + E_2(v_1, v_2, f_1, f_2) \text{ and} \\ f_3(f_1) &= \arg\min_{f_3} H_3(f_3) + E_2(v_1, v_3, f_1, f_3). \end{aligned} \tag{A.20}$$

Now, we recursively solve the subproblems in reverse order (i.e. we solve every subproblem on a level, then go to the adjacent lower levels).

- Level 1 (see Figure A.4a)

$$f_1^* = \arg\min H_1(f_1). \tag{A.21}$$

- Level 2 (see Figure A.4b)

$$\begin{aligned} f_2^* &= \arg\min_{f_2} H_2(f_2) + E_2(v_1, v_2, f_1^*, f_2), \\ f_3^* &= \arg\min_{f_3} H_3(f_3) + E_2(v_1, v_3, f_1^*, f_3). \end{aligned} \tag{A.22}$$

- Level 3 (see Figure A.4c)

$$\begin{aligned} f_4^* &= \arg\min_{f_4} H_4(f_4) + E_2(v_2, v_4, f_2^*, f_4), \\ f_5^* &= \arg\min_{f_5} H_5(f_5) + E_2(v_2, v_5, f_2^*, f_5), \\ f_6^* &= \arg\min_{f_6} H_6(f_6) + E_2(v_3, v_6, f_3^*, f_6). \end{aligned} \tag{A.23}$$

- Level 4 (see Figure A.4d)

$$\begin{aligned} f_7^* &= \arg\min_{f_7} H_7(f_7) + E_2(v_5, v_7, f_5^*, f_7), \\ f_8^* &= \arg\min_{f_8} H_8(f_8) + E_2(v_5, v_8, f_5^*, f_8), \\ f_9^* &= \arg\min_{f_9} H_9(f_9) + E_2(v_6, v_9, f_6^*, f_9). \end{aligned} \tag{A.24}$$

The pseudo-code for one of the possible ways to implement this algorithm is illustrated in Figure A.5.

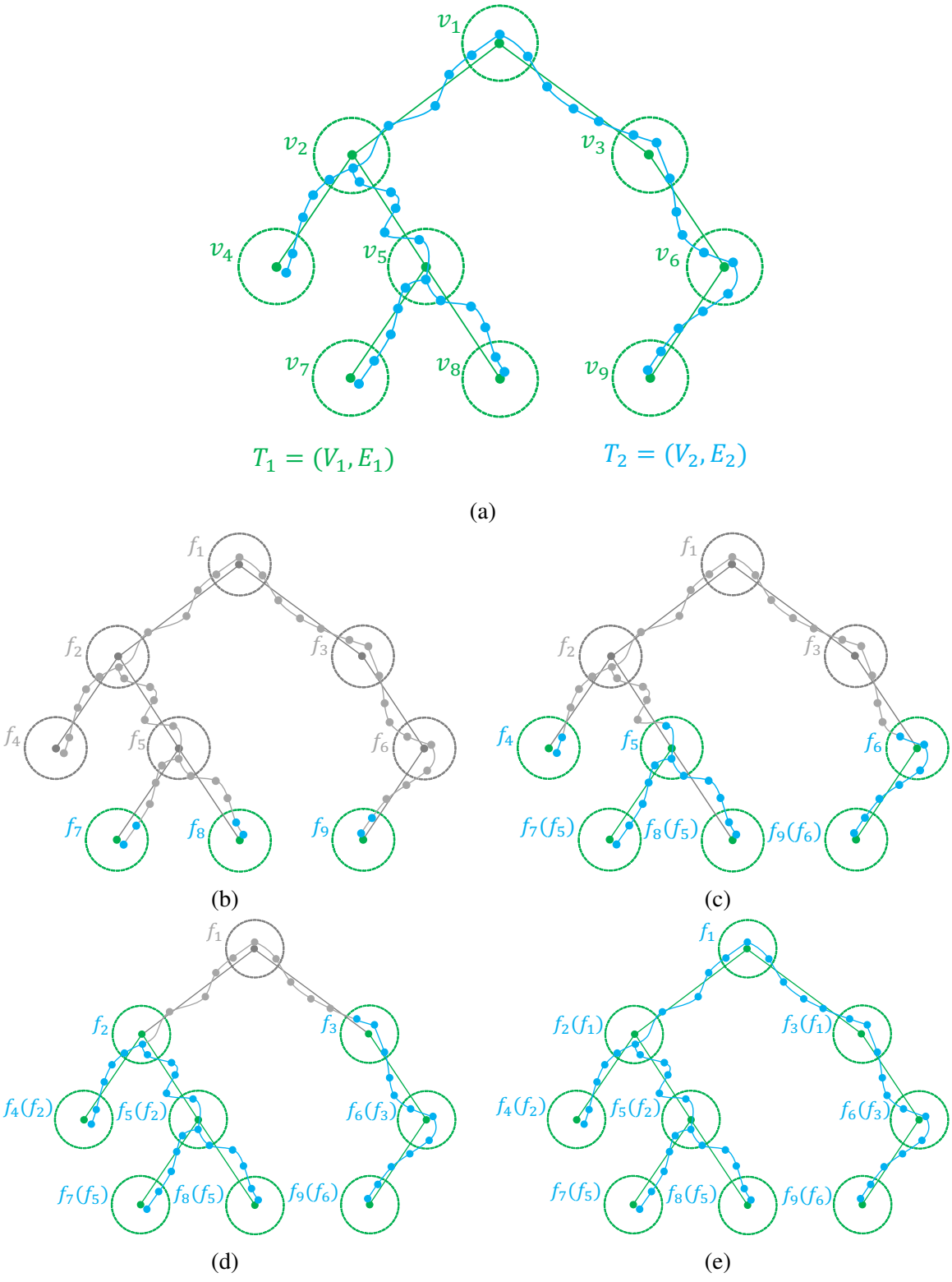


Figure A.3: The Viterbi algorithm for a tree. (a) illustrates the matching problem for ground truth tree T_1 and testing tree T_2. (b-e) subsequent steps of the forward pass of the Viterbi algorithm.

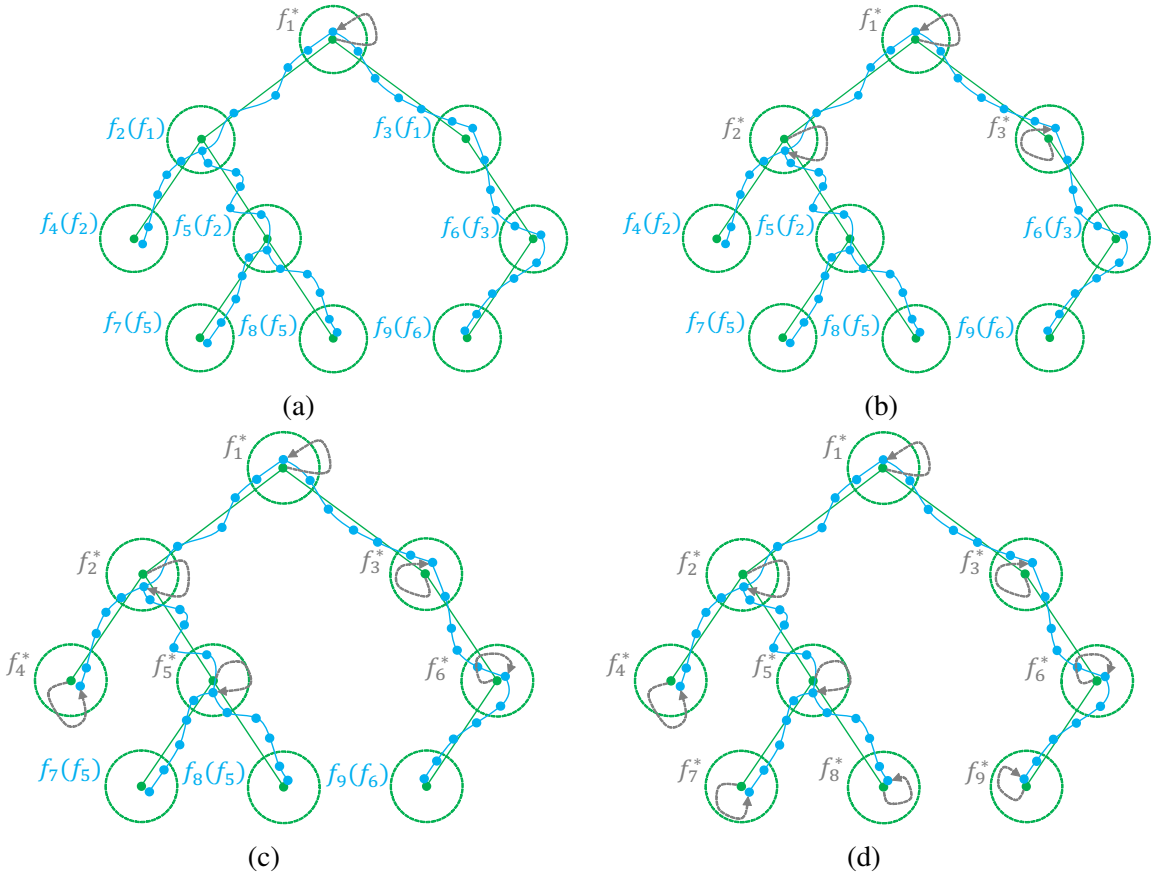


Figure A.4: The Viterbi algorithm for a tree. (b-e) subsequent steps of the backward pass of the Viterbi algorithm.

Input: $v_1, v_2 \ldots v_n \in V_1$
$E_1(v, f(v))$
$E_2(u, v, f(u), f(v))$

for *level* $\boldsymbol{k}$ *starting with the lowest* **to** *the highest* **do**
  foreach *node* $\boldsymbol{v_i}$ *on a level* $\boldsymbol{k}$ **do**
    foreach *label value* $\boldsymbol{f_i}$ *of node* $\boldsymbol{v_i}$ **do**
      $\text{cost}[i][f_i] \leftarrow E_1(v_i, f_i)$
      foreach *adjacent to* $\boldsymbol{v_i}$ *node* $\boldsymbol{v_j}$ *on a lower level* $\boldsymbol{k+1}$ **do**
        $\text{min\_cost} \leftarrow \infty$
        foreach *label value* $\boldsymbol{f_j}$ *of node* $\boldsymbol{v_j}$ **do**
          if $\text{min\_cost} > E_2(v_i, v_j, f_i, f_j) + \text{cost}[j][f_j]$ **then**
            $\text{child\_label}[j][f_i] \leftarrow f_j$
            $\text{min\_cost} \leftarrow E_2(v_i, v_j, f_i, f_j) + \text{cost}[j][f_j]$
          end
        end
        $\text{cost}[i][f_i] \leftarrow \text{cost}[i][f_i] + \text{min\_cost}$
      end
    end
  end
end

for *node* $\boldsymbol{v_i}$ *on the highest level* **do**
  foreach *label value* $\boldsymbol{f_i}$ *of node* $\boldsymbol{v_i}$ **do**
    $\text{opt\_label}[i] \leftarrow$ *any label value* f_i
    if $\text{min\_cost}[i][\text{opt\_label}[i]] > \text{min\_cost}[i][f_i]$ **then**
      $\text{opt\_label}[i] \leftarrow f_i$
    end
  end
end

for *level* $\boldsymbol{k}$ *starting with the highest* **to** *the lowest* **do**
  foreach *node* $\boldsymbol{v_i}$ *on a level* $\boldsymbol{k}$ **do**
    foreach *adjacent to* $\boldsymbol{v_i}$ *node* $\boldsymbol{v_j}$ *on a level* $\boldsymbol{k+1}$ **do**
      $\text{opt\_label}[j] \leftarrow \text{child\_label}[j][\text{opt\_label}[i]]$
    end
  end
end

Result: opt_label

Figure A.5: Pseudo-code of Viterbi algorithm for tree.

Curriculum Vitae

Name:	Egor Chesakov
Education:	**2015** MSc in Computer Science, University of Western Ontario Thesis: *Vascular Tree Structure:Fast Curvature Regularization and Validation* Advisor: Dr. Yuri Boykov **2010** Diploma of Higher Education (with honor) in Applied Math & Informatics, National Research Saratov State University, Saratov, Russia.
Honours and Awards:	**2015** Faculty of Science Graduate Student Teaching Award, University of Western Ontario. **2014 – 2015** Western Graduate Research Scholarship (WGRS), University of Western Ontario.
Certificates:	**2015** Ontario Summer School on High Performance Computing (West), University of Western Ontario.
Related Work Experience:	Graduate Teaching Assistant, University of Western Ontario. **2014 – 2015** Graduate Research Assistant, Computer Vision Research Group, University of Western Ontario. **2014 – 2015** Software Developer, Geofiztehnika, Saratov, Russia. **2011 – 2014** Software Developer, Lapic, Saratov, Russia. **2010 – 2011**