Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS

Ján Dupej

Vessel segmentation

Department of Software and Computer Science Education

Supervisor of the master thesis:	RNDr. Josef Pelikán
Study programme:	Informatics
Specialization:	Software Systems

Prague 2011

I would like to express my gratitude to my consultant, RNDr. Josef Pelikán for the valuable advice and assistance he has provided.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, August 4th, 2011 Ján Dupej

Název práce: Segmentace cév
Autor: Ján Dupej
Katedra / Ústav: Katedra software a výuky informatiky
Vedoucí diplomové práce: RNDr. Josef Pelikán, KSVI

Abstrakt:

V téhle práci je podán přehled některých dostupných segmentačních a visualizačních technik pro angiografii na CT datech. Dále je v práci navrhnut, implementován a otestován systém který umožňuje jak poloautomatickou, tak automatickou segmentaci cév a jejich visualizaci. Pro segmentaci a trasování cév byl použit algoritmus narůstání oblastí vylepšen o několik heuristik a spojen s detekcí středu cévy. Potom byl tenhle algoritmus automatizován pomocí automatického generování počátečných bodů pro segmentaci. Visualizace je implementována jako adaptace známé metody straightened CPR, která byla rozšířena na visualizaci celého průřezu cévy, nikoliv jen jedné čáry na průřezu. Jako další vylepšení byla použita Bishopova soustava souřadnic pro minimalizaci krutu cévy při sledování její průřezu.

Klíčová slova: segmentace cév, analýza medicinských dat, objemová data

Title: Vessel segmentation
Author: Ján Dupej
Department / Institute: Department of Software and Computer Science Education
Supervisor of the master thesis: RNDr. Josef Pelikán, KSVI

Abstract:

In this thesis we researched some of the blood vessed segmentation and visualization techniques currently available for angiography on CT data. We then designed, implemented and tested a system that allows both semi-automatic and automatic vessel segmentation and visualization. For vessel segmantation and tracking we used a region-growing algorithm that we overhauled with several heuristics and combined with centerline detection. We then automated this algorithm by automatic seed generation. The visualization part is accomplished with an adaptation of the well-known straightened CPR method that we enhanced so that it visualizes the whole cross-section of the blood vessel, instead of just one line of it. Furthermore, we used the Bishop frame to maintain minimal twist of the curve-local coordinate system along the whole vessel.

Keywords: vessel segmentation, medical data analysis, volume data

Contents

Introduction

Computed tomography, often abbreviated as CT or CAT scan, is a fairly routine medical examination these days. A three-dimensional image of the scanned subject is generated by measuring the attenuation of X-ray radiation passing through the subject. Even a simple scan yields massive amounts of data that must be efficiently processed and transformed to something that is humanly readable and understandable. To that end we need software that can work with such data.

Probably the simplest way of analyzing this three-dimensional (3D) volumetric data is by examining the individual slices. This approach is still preferred by many physicians, however proper diagnosis might be increasingly difficult in the case of veins. The cause of this problem is mainly that veins (even when injected with contrast agents) appear as very small circular or elongated objects, depending on their orientation, and it is therefore difficult to make a statement as to the physical dimensions or the general spatial location of the vessel based only on one visible slice. Computer graphics methods may, however, alleviate this problem considerably.

There has been some work done in these areas. The results include automatic or semi-automatic segmentation of these organs with respect to aggravating factors such as noise in captured images. Furthermore, there are visualization techniques that will effectively display the vessel as it would appear laid on a straight plane or even stretched to a linear shape. In this thesis we will design and implement algorithms for such segmentation and visualization, trying to improve on existing methods. We will then compare our results to some of the existing visualization methods.

1 Computed Tomography

The first laboratory CT scanner has been built in 1967 by Godfrey N. Hounsfield at the Central Research Laboratories of EMI Ltd. The specimen was scanned from 360 uniformly distributed directions. The data acquisition took whole 9 days, because of the low intensity of the used gamma radiation source. After that, the 3D image had to be reconstructed. This took about 150 minutes on a period computer. (1)

Since then, significant improvements have been made resulting in the first clinically available device, installed at Atkinson-Morley Hospital in September 1971. Image acquisition time was about 4.5 minutes, which is rather lengthy by today’s standards, however significantly faster than Hounsfield’s prototype. The first patient was scanned by this device on October 4th, 1971. (1)

Independently of Hounsfield, Allan M. Cormack derived the mathematical theory for image reconstruction and did other significant work in the area. Their efforts were recognized in 1979 with a Nobel Prize in Medicine. (2)

1.1 Basic Principles

CT uses an X-ray beam that passes through a thin axial section of the scanned subject. The X-ray tube rotates around the patient irradiating them with a collimated fan-shaped beam. After passing through the subject, the beam hits the detector array. While the 1970s scanners used one or a few detectors, the current devices use a whole array of detectors arranged in a full ring (4th generation) or an arc (older 3rd generation) around the patient. (3) This allows for a much shorter scan time (up to 300 seconds for the 1st generation, and up to 5 seconds for the 4th generation) and strain on the patient, not to mention less exposure to radiation.

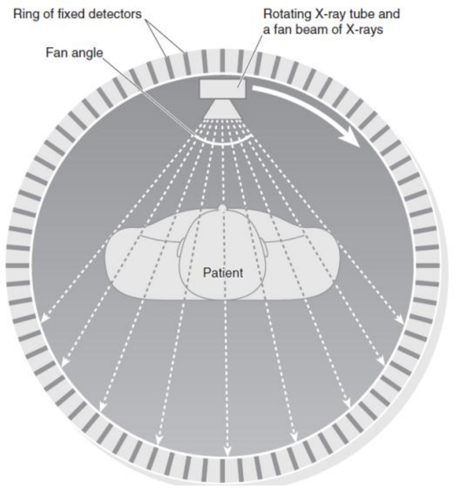


Figure 1: Schematic of a fourth generation CT scanning (cross-section) (picture from (4))

Further attempts to speed up the imaging process include the spiral CT, electron beam CT or the multi-slice CT.

1.1.1 Image Reconstruction

The output the detectors provide are basically the signal attenuations along the paths from the X-ray tube (from many positions around the subject), through the patient to the detector array. This data, after various corrections made to compensate for some hardware-specific errors (like inhomogenities of the detector assembly) is called the *CT raw data* (3).

We, however, want to reconstruct the attenuation of the scanned material in uniformly distributed points on the slice from the total attenuations observed from the side. A solution was first developed in 1917 when Radon solved the problem of reconstructing a function from its line integrals. (1)

Let us assume that the X-ray photons are all of the same energy, the intensities of the ray entering and leaving a uniform material conform to the Beer-Lambert's law:

$$I = I_0 \cdot e^{-\mu \cdot \Delta x}, \tag{1}$$

Where I_0 is the intensity of the entering ray, I is the intensity of the ray as it leaves the material, Δx is the thickness of that material and μ is its linear attenuation coefficient. (1)

As the scanned object is non-uniform, we will assume that it is made up of a finite number of equally large, homogenous parts. We may then rewrite the attenuation equation to:

$$I = I_o \cdot e^{-\mu_1 \cdot \Delta x} \cdot e^{-\mu_2 \cdot \Delta x} \cdot \dots \cdot e^{-\mu_n \cdot \Delta x} = I_o \cdot e^{-\Delta x \cdot \sum_{k=1}^{n} \mu_k} \tag{2}$$

Where μ_k are the individual attenuation coefficients for each of the homogenous parts. It is then advantageous to divide both sides of the equation by I_0 and take a logarithm of them. We will then have a quantity that is a sum of those attenuation coefficients.

$$p = -ln\left(\frac{I}{I_0}\right) = \Delta x \cdot \sum_{k=1}^{n} \mu_k \tag{3}$$

This alone inspires a naïve solution to the problem of extracting the individual attenuation coefficients. Supposing we want to extract a 2x2 grid of such coefficients from their projections as shown in Figure 2, we need to solve a system of linear equations.

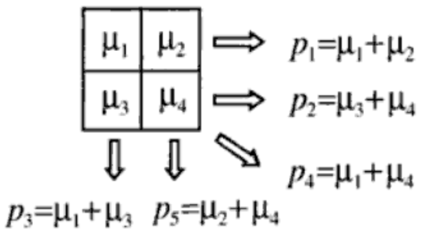


Figure 2: Sample attenuation coefficients and their projections (picture from (1))

This presents us with several problems, not the least of which is that errors in the raw data may throw the calculations off. Furthermore, when the number of measurements exceeds the number of unknowns, a straightforward solution may not be available (1).

The actual method uses a filtered back-projection formula that involves 2D Fourier transforms. The derivation of this formula is beyond the scope of this thesis; however Hsieh (1) gives a solid mathematical background for CT image reconstruction.

1.1.2 The Hounsfield Unit

The output of a CT scanner is the local density in a set of spatial points. Medial scanners are calibrated to the Hounsfield Unit (HU) scale. Each medical CT will give the same value for the same kind of tissue. The densities on that scale range from -1024 to 3071 HU (4). Some of the typical materials and their densities are listed in Table 1.

Material	**Typical density (in HU)**
Air	-1000
Water	0
Fat	-120
Muscle	40
Blood	30 to 45
Contrast agent	130
Bone	800 to 1200

Table 1: Typical Hounfield densities of some select materials (5)

The HU value of a particular material can be calculated using its linear attenuation coefficient, and such coefficients of air and water. The formula is simple:

$$HU = 1000 \cdot \frac{\mu_X - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{4}$$

Where μ_X, μ_{water} and μ_{air} are the linear attenuation coefficients of the unknown material, water and air, respectively.

1.1.3 Processing the Data

One series of data from the scanner contains a great deal of information. Typically, examining such data is more complicated than the more straight-forward look at an X-ray image. Medical examinations involving such data can be assisted with appropriate data processing. Typically, the necessary tasks can be broken down into the following groups:

- *Registration* is mapping two or more images onto each other, so that they are matched somehow. Spatial matching is the most common. It means that each corresponding pair of points on both images represents the same scanned location.
- *Segmentation* splits the image into semantically corresponding areas, like finding the areas which capture the vessel and those that capture anything but a vessel. Segmentation is probably the most commonly performed task, as it often constitutes the first step of more complex methods.
- *Classification* decides whether a segmented area in an image is likely to be of some defined kind.
- *Reconstruction* generates the geometric representation of captured objects.

1.2 Angiography with CT

Conventional angiography is the medical examination method which involves injecting iodine-based contrast agent directly into the examined blood vessel and scanning with an X-ray scanner. An enhanced image can be obtained by digitally subtracting the mask obtained before the application of contrast agent from the images with contrast. This reduces the amount of irrelevant information in the image and makes the method widely accepted and used. The advantage of this conventional angiography is the possibility of immediate medical intervention when needed. On the other hand, it is rather invasive and uncomfortable for the patient (2).

Angiography is the medical examination of veins. CT angiography (CTA) has provided many improvements in vascular imaging. The main advantages over conventional arterial angiography include significantly lower invasiveness, less radiation exposure, lower cost and better patient response. From a medical point of view, CTA provides better diagnostic options, including viewing the vascular anatomy from multiple angles or simultaneous visualization of vessel wall and lumen. (3) CTA can be combined with several visualization techniques.

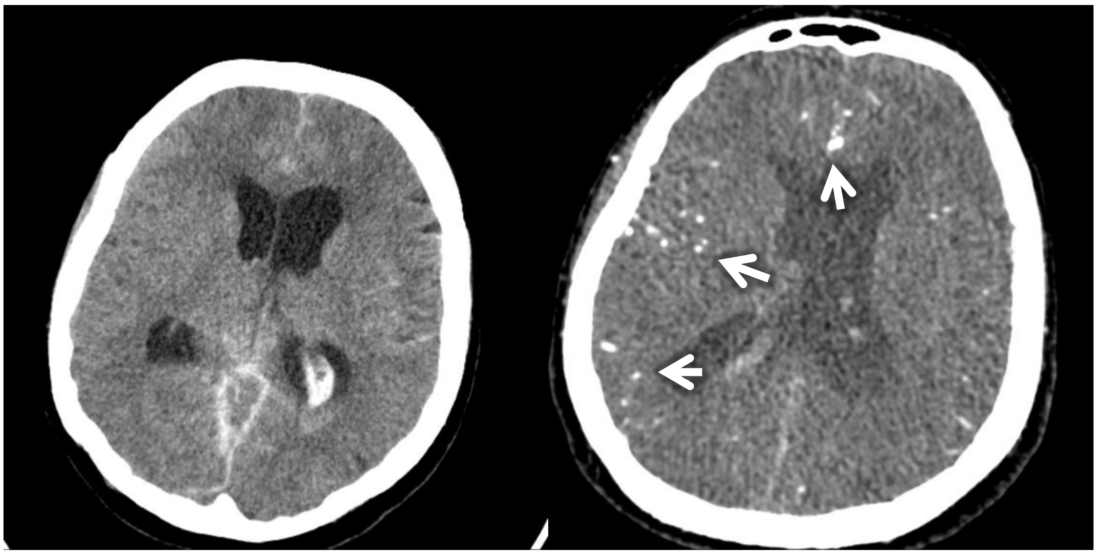


Figure 3: A CT-scanned cross-section of human head. Left - without contrast, Right - with contrast (note the visible opacity-enhanced veins)

As it can be seen in Table 1, the density of blood is very close to the density of muscle tissue. For this and other reasons, it is important to inject the patient with a contrast agent. Generally, small veins require more opacification. Less of an enhancement is required for large vessels (e.g. the thoracic aorta) and vessels that are perpendicular to scanning planes (e.g. carotid artery) (3). Typically, iodine and barium are used for CT (6).

2 Vessel Segmentation

Segmentation is an important part of medical volumetric data analysis. There is a host of various algorithms for different organs and scanning modalities. Unfortunately, there is no single algorithm that can extract vasculature from every image modality (7). Various algorithms that use either intensity-based pattern recognition or vessel contours have been designed.

In addition to that, there has been some work done in segmentation techniques based on artificial intelligence and neural networks (7). They are used to simulate biological learning processes mostly for classification. We will, however not include them in our overview as they are of less relevance for our application.

This chapter explains some of the challenges faced by vessel segmentation algorithms and then gives an overview of existing approaches based on both pattern recognition and model-based techniques.

2.1 Challenges

There are many caveats in designing a viable vessel extraction algorithm. This subchapter gives an overview of the most severe ones.

Volume data from both CT and magnetic resonance imaging (MRI) is plagued by *noise*. The amount and type of noise differs between various scans and modalities. Moreover, in CT there are artifacts caused by very dense materials, such as metals or even teeth. Dental fillings or metallic joint replacements can cause visible streaks, otherwise known as *beam hardening* (4). Motion blur is inevitable since the scan can take a few seconds, which is more than enough time for effects such as heart motion or patient movement to disrupt data reconstruction.

A CT scanner has a finite resolution. The organ boundaries, namely their different densities captured within a single voxel cause the averaging effects, known as *partial voluming* (4). This may be especially problematic for vessel segmentation, as vein calcifications have a density very similar to bones. Therefore if such a vein is in a close proximity to a bone, the boundary between the vein and the bone may be virtually indistinguishable. (2)

Last, but hardly the least, there is a large *variability* in the captured data. This is caused by natural physical variances in the patients; shape of segmented organs or even by different scanning set-ups (e.g. the patient tilting his head on the scanner gantry, scanner configuration). CT angiography makes use of contrast enhancing agents. Even their distribution in the visualized vessels may differ, causing potential difficulties for segmentation algorithms.

A good segmentation method should, of course, be able to deal with all those challenges. However, to author's knowledge, there are no such algorithms at present.

2.2 Pattern Recognition Methods

Pattern recognition (PR) methods implement automatic detection of objects or features. Humans do this very well; even some of the methods are adaptations of human-style PR on a computer. The methods described in this section include the region growing algorithms that focus on extracting the center line of the vessel and on approaches exploiting mathematical morphology. Furthermore, rationale for performing segmentation in multiple scaled versions of the source image is given.

2.2.1 Multi-Scale Approaches

As the subsection caption may hint, multi-scale methods perform segmentation at varying image resolutions. As the segmentation may be slow due to the sheer size of the data, processing it at a lower resolution can be helpful in optimizing speed. Dealing with the large structures at low resolutions while processing the smaller objects at a higher resolution potentially increases algorithm robustness. The image may also contain substantial amounts of noise. If that noise is mostly of high frequency, scaling the image down may increase the algorithm's resistance to such noise. One important consideration with multi-scale techniques is the proper choice of scales in which the algorithm will work.

A typical multi-scale approach application is demonstrated by Sarwal and Dhawan (8). They reconstruct 3D coronary arteries from three views by matching image features, vessel branch points in their work. For increased robustness, their matching process is performed at three different resolutions. Larger vessel tree branches are processed at lower resolution, while the thin ones are extracted at a finer scale.

2.2.2 Centerline Detection Algorithms

These methods, otherwise known as skeleton-based extract the center of blood vessels. Then a vessel tree is created by connecting these centerlines. Many different methods are used to calculate the vein centerline.

For instance, Sorantin at al (9) use a sort of 3D skeletonalization method for assessing tracheal stenoses on spiral CT data. Their method consists of several steps, including seed-initialized segmentation of laryngo-tracheal tract (LTT), followed by conversion of the segmented object into isotropic cubic voxels and 3D thinning. The medial axis is then extracted using a shortest-path searching algorithm and low-pass filtered to achieve a smoother curve. After that, cross-section profiles are calculated along that centerline. That technique is reported as very accurate and precise on their phantom studies. (7)

2.2.3 Region Growing Algorithms

The region growing algorithms segment objects by incrementally adding new points to the region. Operating under the assumption that points with similar intensity that are close together belong to the same object, the seed region is appended with new points in its proximity as long as they meet the homogeneity criteria.

Noisy data can be a problem for region growing algorithms as it may result in holes or over-segmentation. Therefore, some post-processing is definitely called for, even though there is an enhancement called adaptive region growing, presented by Yi and Ra (10) which is more resilient to those problems. This method basically performs locally adaptive segmentation within small local cubes. The faces of the local cubes have vessels entering and leaving through them. Region growing is performed only within the cubes that are then connected to one another depending on whether a vessel passes through the adjacent faces of two cubes. This way, a vessel tree can be created. Their method works on both CT and MRA (magnetic resonance angiography) data.

2.2.4 Vessel Tracking by Calculating Minimal Path

Minimal path algorithms like Dijksta or Floyd-Warshall can be used to track vessels if we convert the volume data into a graph. Such conversion is not difficult; each voxel of the volumetric data is considered a vertex of the graph and is connected to its six immediate neighbors (four, if we work in 2D) by edges.

One example of algorithms utilizing calculating the minimal path in a graph is the live wire introduced by Falcao in (11). His method estimates the vessel boundary, which is assumed to be a closed, contiguous curve consisting of directed edges with weights, which are named *bels* (boundary elements). The method can operate in 2D as well as 3D. The weight of each bel is calculated from some of its other properties, for instance the intensity gradient of voxels adjacent to that bel. After the graph is constructed, the path between two user-selected points is found by Dijkstra's algorithm.

Significant speed enhancements to this method have been presented by (12) and (13). The latter describes the intelligent scissors, a method that can significantly confine the volume in which the minimal path is sought, providing a considerable speed boost.

2.2.5 Mathematical Morphology

Morphology can be understood as the study of shapes. Morphological operators (MO) apply transformations to images using structuring elements (SE). These methods are typically applied to binary images, however extensions to grey-level images are normally available. The two main MO are erosion and dilation. Erosion shrinks objects by a SE, while dilation expands them and does other work like filling holes or merging disjoint regions. The most interesting mathematical morphology-related algorithms that are applicable in segmentation are the top-hat and watershed transformations.

The watershed algorithm is indeed well named. Its basic principle can be explained on the analogy of pouring water into a landscape or a topological surface defined by the input image. The image corresponds to a height-map of the surface. The water accumulates in basins, generally forming a growing region around the low points, avoiding the high points. A watershed algorithm on grey scale images with applications in angiograph segmentation is proposed by Couprie and Bertrand in (14).

2.3 Model-based Approaches

The model-based approaches attempt to match a predefined model to the image. In this section we will give an overview of deformable models and template matching.

2.3.1 Deformable Models

One of the most well-known methods belonging to the deformable models class are the active contours, also called snakes. It is basically an optimization of a closed curve defined by its points (snaxels). Several kinds of forces deform the curve. First, the curve is held together and smooth by internal forces. Second, external forces try to match the curve to the shape we are trying to segment (often defined as gradients of image values). Additional forces that make the curve obey some other constraints may also be defined. Speaking formally, fitting a snake to the desired shape is equivalent to finding the parametric curve $\mathbb{v}(t)$ that minimizes (15):

$$E^*_{snake}(\mathbb{v}) = \int_0^1 E_{snake}\big(\mathbb{v}(t)\big) \cdot dt \tag{5}$$

Where E_{snake} is the energy of the snake for a particular curve, which is basically the sum of the internal forces E_{int}, external forces E_{image} and constraint-enforcing forces E_{con}.

$$E_{snake}(\mathbb{v}) = E_{int} + E_{image} + E_{con} \tag{6}$$

Optimizing a snake can be done by various methods as well. An ingenious method was proposed by Geiger et al (16) that detects tracks and matches deformable models using a dynamic programming approach, but unlike most other methods is non-iterative and guaranteed to find the minimum snake. A detection algorithm is used to generate a list of uncertainty points for each seed. After that a search window is created from two consecutive lists. The optimal contour passing through the two lists is calculated using a dynamic programming-based algorithm that considers all possible contours and their deformations. Dynamic programming is inherently slow and memory-consuming, therefore multiscaling is used to balance speed and curve optimality. When contour tracking finishes on one frame, the same contour is used to seed the contour matching on the next frame.

A comprehensive review of active contours and their enhancements is presented by Krajíček in his mater thesis (15).

2.3.2 Template Matching

These methods work by recognizing a structure model (template) in the processed image. For the purposes of vessel segmentation, the arterial tree template is normally represented by a series of nodes connected in segments. That template is then deformed to match the structures in the scene.

One of the simpler template matching methods applicable in angiography is the ellipse fitting (17). Intuitively, the algorithm tries to find the best way to match an ellipse to the image. From those parameters the center of ellipse is calculated and used as a point on a vessel walk path.

First, the image is processed with Canny edge detection filter to find the set P. Then, the algorithm searches for the optimal conic section $C(\mathbb{a}) = \{x | F(\mathbb{a}, x) = 0\}$ to fit the set P. Simply put, we want such a vector $\mathbb{a}$ for which all the points of P lie in $C(\mathbb{a})$. A cone section is defined by the equation

$$F(\mathbb{a}, x) = \mathbb{x}_i \cdot \mathbb{a}^T \tag{7}$$

Where $\mathbb{x}_i = \left[x_i^2, x_i y_i, y_i^2, x_i, y_i, 1\right]$. Solutions to this problem are given by Fitzgibbon and Fisher (17) and by Halíř and Flusser in (18). The former gives a direct approach to finding the parameters of the optimal ellipse with least-square fitting method. The latter work even provides a numerically stable, non-iterative method for finding an optimal $\mathbb{a}$. Both methods are fast and have a good resistance to noise.

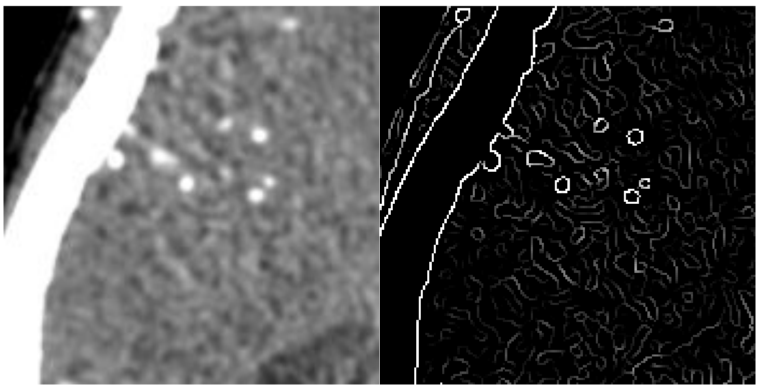


Figure 4: Left - part of head CT cross-section; Right - same image with Canny filter applied[1]

The obvious problem with ellipse fitting is the Canny filter. It is necessary to use the filter properly, so that it finds points on the circumference of the vein cross-section only. As illustrated by Figure 4, the filter, as it is, does find edges, but not solely of veins. Therefore using this method on real-life data may be problematic due to limited robustness.

[1] Filtering performed online with http://matlabserver.cs.rug.nl/cgi-bin/matweb.exe

3 Vessel Visualization

As this thesis is mainly oriented on segmentation, we will only give an overview of the Curved Planar Reformation (CPR) and three of its flavors as introduced by Kanitsar at al in (19). This method has been developed for visualizing veins and duct-like objects; therefore we will use it to visualize our results of segmentation in the next chapter, albeit in a slightly extended form. The CPR algorithm family basically displays the vessel within its small neighborhood. This dramatically reduces clutter and thus enhances the diagnostic possibilities.

3.1 Curved Planar Reformation

Visualizing vasculature in volumetric data obtained from CT scanners can be difficult, considering there is a lot of data of less interest. The typical rendering techniques like ray casting or maximum intensity projection (MIP) would be impractical due to extensive configuration requirements needed for proper visualization of veins. CPR has been developed precisely for that purpose (19). This method visualizes the whole length of the tubular structure in a single image. In this section, we will examine the three variants of this method. Each of them has different properties, like length preservation or spatial perception level. All of them, however, share a common prerequisite - the knowledge of vein centerline. A discretized representation as a list of points, preferably at sub-voxel resolution, will suffice.

The methods visualize the surface that contains the vein centerline. In order to precisely define the surface, we need a vector of interest. A line-of-interest is defined by a point on the vessel centerline and the vector-of interest. The voxels intersected by that line are taken to form the visualization.

3.1.1 Projected CPR

The projected CPR is basically a projection of the data set, taking into account only the thin slice of voxels that are intersected by riding the line-of-interest along the vessel centerline.

Without loss of generality, let us assume the vector-of-interest is collinear with the y-axis. Then projected CPR will do a parallel projection of the free-form surface along the x-axis. It may, of course, happen that the path of the centerline defines will make the line-of-interest pass one point in the projection space several times (e.g. a longer part of the vessel running along the x-axis or making a U-turn). In this case, the values are not simply overwritten, as this would cause some information loss. Instead, pixels are composited using a maximum intensity projection or averaging. These two methods are perhaps the most widely used visualization techniques for medical volumetric data. We will give a short peek into how they work in the fifth chapter.

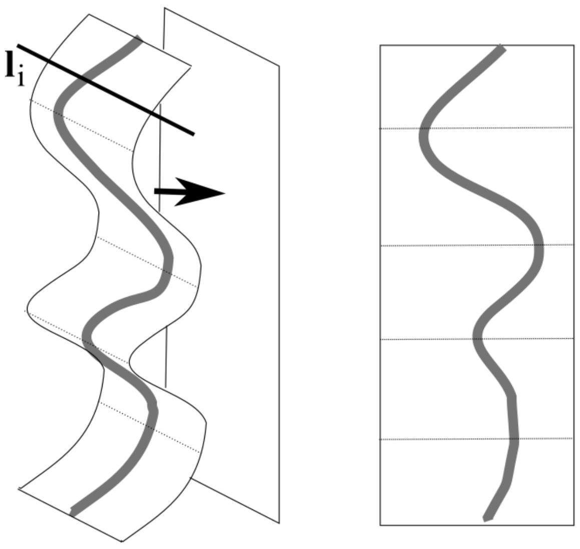


Figure 5: Projected CPR: Left - vessel in 3D with the line-of-interest and the sampling plane described by those lines; Right - the visualization generated by projected CPR

As a result of the parallel projection, spatial relations are still preserved in this projection. On the other hand, sections of higher intensity (bone) may occlude other lower-intensity sections (blood vessels), rendering them invisible. In addition to that, isometry is not preserved, which means that the real length of the vessel will appear distorted, especially if the vessel ever travelled perpendicularly to the projection plane. In addition to that, the vessel may be rendered incompletely, producing an image that very much resembles a stenosis, even in areas where the vessel is dilated due to the presence of an aneurism.

3.1.2 Stretched CPR

The surface defined by the lines-of-interest is curved in one dimension and planar in the other. Stretching that curved dimension into a plane will show the tubular structure without any overlapping. This is the main difference between projected CPR and stretched CPR.

The curvature of the vessel is still mostly maintained. This method has another advantage as compared to projected CPR, and that is maintained isometry. The visualized vessel will appear curved, but despite that the actual length of the vessel can be estimated from this projection. However, just like with the previous member of the CPR family, smaller calcifications are likely to be hidden as a result of visualizing only one line-of-interest. Another undesirable feature that is shared with projected CPR is the possible incompleteness of vessel projection, which may mimic stenosis even in the areas where the vessel is widened.

3.1.3 Straightened CPR

This type of CPR completely straightens the vessel and generates a linear representation with a varying diameter. As opposed to the other two methods, the line of interest is no longer necessarily parallel to axial slices. At each point, the tangent vector $\mathbb{t}_i$ of the curve is calculated. Local coordinate system at a point on the curve is defined by two perpendicular vectors $\mathbb{u}_i$ and $\mathbb{v}_i$, (vessel normal and binormal, respectively) such that $\mathbb{u}_i \perp \mathbb{v}_i$ and both $\mathbb{u}_i, \mathbb{v}_i \perp \mathbb{t}_i$. The vector-of-interest lies in the plane defined by $\mathbb{u}_i$ and $\mathbb{v}_i$ and can be expressed as

$$\mathbb{l}_i = \mathbb{u}_i \cdot cos\varphi + \mathbb{v}_i \cdot sin\varphi \tag{8}$$

Where φ is the parameter of this projection. The local coordinate system is also illustrated in Figure 6.

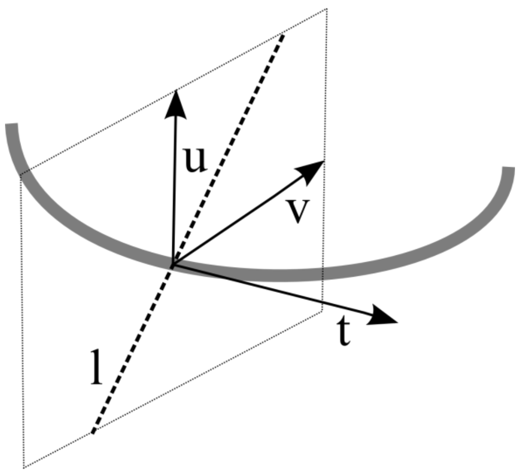


Figure 6: Local coordinate system on the walk along a vessel, showing the line-of-interest

The most obvious disadvantage of this projection is the loss of spatial orientation which may occur due to complete straightening of the vessel. However, straightened CPR preserves isometry. Another serious edge this method has is the clear visualization of the vessel's diameter. This allows for simpler detection of stenosis and the assessment of the vein's cross-section size.

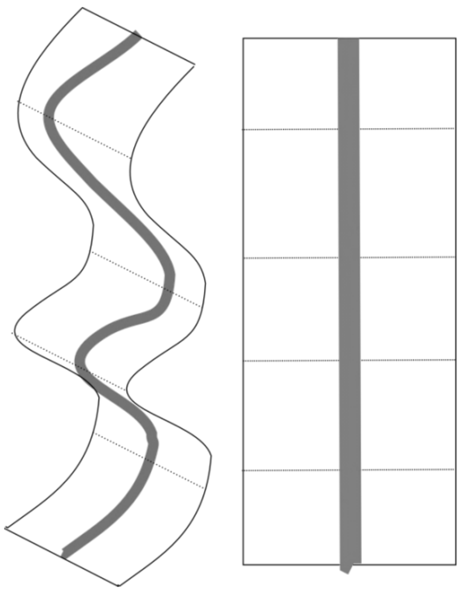


Figure 7: Straightened CPR: Right - the linearized curve

The CPR as presented by (19), however, gives no information as to calculation of the vectors $\mathbb{u}_i$ and $\mathbb{v}_i$. This is a problem that must be solved before using this method. A viable solution may involve using either the Frenet-Serret coordinates or the Bishop frame (20), further analyzed and used later in the thesis.

4 Designing Automatic Segmentation

In this chapter we design a semi-automatic vessel segmentation algorithm and then expand it to a fully automated method. Furthermore, we will derive a visualization method based on existing CPR algorithms that will provide a potentially more precise visualization of the segmented blood vessels. Some details of the implementation including the interfaces required to extend our program with new segmentation or visualization algorithms are given in Appendix A. Additionally, Appendix B contains a brief overview of the program's functions.

4.1 Semi-Automatic Segmentation

In this section we will design a semi-automatic segmentation method for contrast-agent-enhanced blood vessels from CT data. This method will require seed points to start from. Basically we will use a region-growing algorithm on the individual slices and merge the results into a contiguous vessel sub-tree after processing each slice.

Speaking more formally, the input of this part is the actual volume data and seed point coordinates. The algorithm outputs a vessel sub-tree composed of connected vessel centerlines.

4.1.1 Region Growing Algorithm

First, the region is initialized to a one-point set containing only the seed. We then use region growing only on the individual slices. A modified four-way single pixel recursive flood fill is utilized. The modification was necessary in order for the algorithm to cope with noisy data and involves allowing the flood fill to fill pixels whose value falls within some small interval centered at the seed point's value.

We also use two heuristics to prevent over-segmentation. Before even starting the region growing, we sample the seed point's value. If it does not fall within the expected radio density interval that would indicate a blood vessel with a contrast agent, we search a small 3-pixel neighborhood of that pixel. If none of the checked pixels comply with our criteria, the seed is rejected. As a result, the seed does not even need to be very precise, which may be useful for the automated seed generation, described later in the thesis.

Obviously, we are trying to satisfy two mutually-exclusive conditions here. We want to prevent over-segmentation, yet we want our algorithm to reliably find the vessel. By constraining the threshold for vessel detection we can prevent the region growing algorithm from filling a greater area than it should, but with increasing noise the algorithm will provide more and more discontinuous regions. To take care of that problem, we left the vessel intensity interval rather narrow and implemented a hole-filling algorithm that will eliminate spurious holes in the segmentation results on data with higher noise levels. The hole-filling algorithm simply finds the first and last positively detected pixel in each row and column and fills a solid line between these two points. Having a solid mask of the segmented region also helps the center detection algorithm in the next paragraph.

4.1.2 New Seed Extraction

Once we are done segmenting a slice we need to extract the point that represents this vessel cross section. We chose that point to be the center of mass, which can be calculated very quickly and has proven to match the vessel's actual center well:

$$\mathbb{s} = \frac{1}{|P|} \cdot \sum_{\mathbb{p} \epsilon P} \mathbb{p} \tag{9}$$

Where P is the set of points that the former segmentation step marked as belonging to the vessel. This center point is stored. As the next step we need to come up with the seed points for the segmentation on the next slice. We use the center, top center, bottom center, left center and right center points, in this order, as shown by Figure 8. All these seeds are passed to the region growing algorithm.

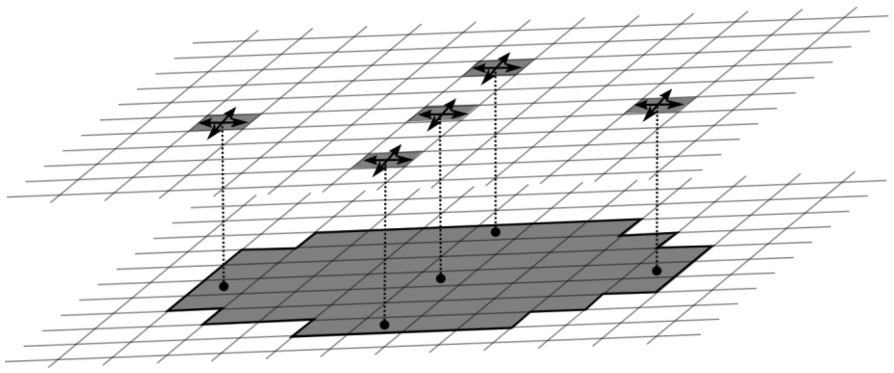


Figure 8: Extracting seeds for the new slice

Those seeds whose value (on the next slice) falls outside the expected interval for contrast-enhanced blood vessels are discarded. If the flood-fill yields different masks for different seeds, the vessel is assumed to have split into as many branches as there were unique masks. If none of the seeds' values indicate a blood vessel, the path is terminated at this point.

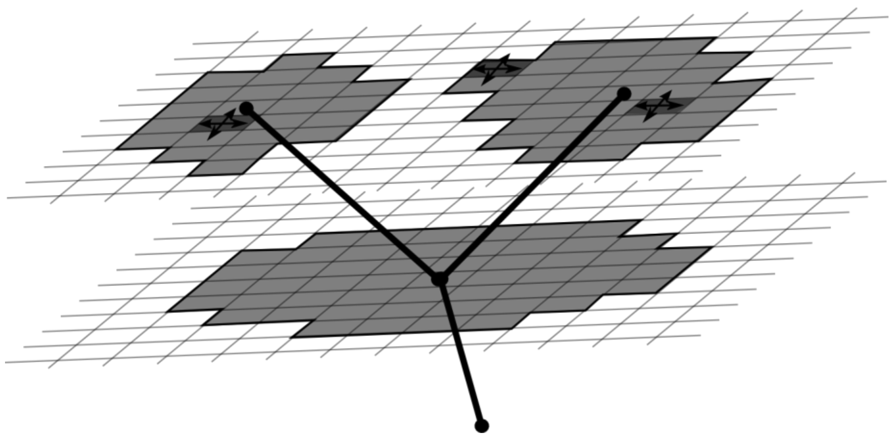


Figure 9: Vessel split detected

For the very first slice we have segmented, we have no information in which direction to continue the segmentation. For that slice, we attempt segmentation on both previous and the next slice (if applicable) – simply put, we assume the vessel to go both ways. With the next slices, we proceed only in their respective directions as they are known at this point.

4.2 Extending to a Fully Automated Mode

At this point, we have a viable method to extract a part of the vessel tree given a point contained within the vessel. However, we want to detect all the vessels within our dataset. One option to accomplish this is to have the user select appropriate seed points for us. This task is indeed tedious as it involves more than just clicking on vessels, not that clicking on several dozen vessels would not be impractical enough. The user must also configure appropriate transfer function or HU window in order to see the veins. Here, we would like to present a method that reduces the process to one click and waiting a few seconds.

The way we are going to do this is by combining the semi-automatic segmentation method described in section 4.1 with an algorithm that sweeps the volume data and generates good seed points for the segmentation engine. Processing speed is also a consideration; therefore we want our seed generator to be simple enough, yet able to reliably detect vein cross sections in the volume data slices.

4.2.1 Filtering and Multiscaling Overview

Even though obvious, it is important to note that our seed generator is run before the semi-automatic segmentation algorithm, therefore it is dealing with raw non-segmented volumetric data. The seed generator parses the volume data in slices, in fact every sixth slice is processed. We justify this optimization by the fact that if a seed point indeed causes the segmentator to find a vessel, it is typically significantly longer than 6 slices.

The pipeline is relatively simple. In a nutshell, we first apply a Gaussian blur and an edge enhancing filer, namely Sobel. Exploiting the fact that vein cross-sections will appear mostly as circular or elliptic objects, we apply a Hough filter for circles. Following that phase, we have the “neighborhood circleness” of every point in the slice. We then find the points that satisfy the minimal “circleness” and cull away the clusters of points by means of local maximum-detecting algorithm that rejects points in close proximity to one another.

Because the Hough transformation searches for circles of a particular diameter, it may have a lower and less localized response for circles that are of a different diameter. We therefore use the input slice in three scales (original size, 2x shrunk and 4x shrunk). This should give us a more adequate response for vessel sizes ranging from small (a few voxels in diameter) to large ones. The seeds from all three pipelines are then combined and duplicate values are removed.

4.2.2 Edge Detection

For edge detection we use a Sobel filter preceded by a Gaussian kernel filtering. Gaussian blurring is simply a convolution (21) of the original image with the appropriate kernel, for our purposes we use the following kernel:

$$\mathbb{G} = \frac{1}{159}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

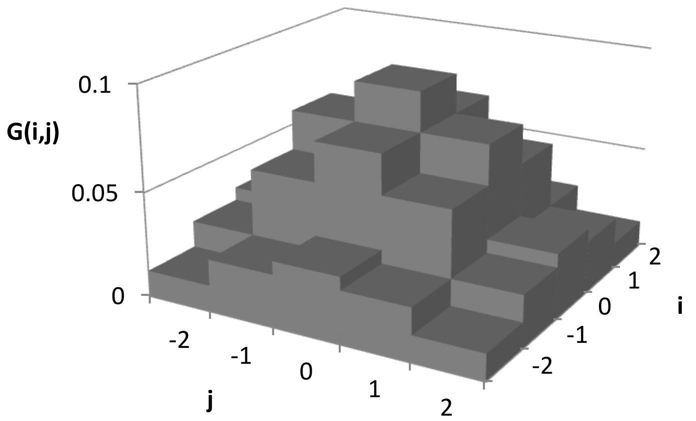


Figure 10: The Gaussian kernel impulse response used for blurring

The Sobel operator (22) is used next. Its output is the approximation of the image gradient at each point. We need only the magnitude of that gradient. To that end we estimate the horizontal $\mathbb{D}_x$ and vertical $\mathbb{D}_y$ derivative of the image values:

$$\mathbb{D}_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} * \mathbb{A};\ \mathbb{D}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix} * \mathbb{A} \tag{10}$$

Where $\mathbb{A}$ is the matrix of original image values. We then combine these two derivative estimates to acquire the gradient magnitude estimate $\mathbb{D}$, for all i, j indices of rows and columns of the image.

$$\mathbb{D}(i,j) = \sqrt{\mathbb{D}_x^2(i,j) + \mathbb{D}_y^2(i,j)} \tag{11}$$

4.2.3 Hough Transform for Circles

After we are done enhancing the edges of the image, we run the well-known feature extraction method, the Hough transform, specifically its modification for finding circles. The Hough transform uses an accumulator array; its dimension is equal to the number of unknowns. In our case, we are interested in the position (x and y coordinates) of the circles of a particular diameter in the image. To that end a voting procedure is run. There are many papers on available, for instance (23) presents a voting scheme for efficient detection of lines and extends that on ellipses as well or (24) designs a method for recognizing circles of various radii.

The algorithm normally processes the image space and updates the voting array appropriately when an edge (from previous edge detection) is hit. We do practically the same process, optimized by discretizing the searched circle. Every time our implementation hits a potential feature (edge) we update the cells of the voting array defined by the offsets in the pre-computed circle array.

Even though the Hough transform for circles can be easily extended to detect circles of various diameters, due to robustness and performance reasons we have opted to detect circles of only one diameter, but on three down-sampled versions of the image.

4.2.4 Seed Extraction

The seed extraction algorithm operates in scan-line mode and uses one seed time-to-live (TTL) buffer for rejecting swarms of closely-packed positive detections. The idea is simple: For every line find the local maxima of image values and consider them potential seeds. To get rid of spurious false positives, we also apply a quick thresholding. Points that are local maxima, but the Hough transformation response at them was too low are ignored.

Now, we try adding these seed candidates to the definitive seed list one by one, marking a small neighborhood (`seedRadius` pixels on either side) of that point as inhabited in the TTL buffer (for `seedTTL` next lines). When attempting to make a seed definitive, check the appropriate cell of the TTL buffer. If it is already inhabited, discard that seed. After each line, decrement every cell of the TTL buffer by 1 (and clamp to zero). For details, see the pseudo-code in Figure 11.

```
function ExtractSeeds(ref List<Point> finalSeeds)
InitializeArray(ttl, 0)

for y = 0 to imageHeight
   for x = 1 to (imageWidth – 1)
      if imageValue(x-1,y) < imageValue(x,y) >= imageValue(x+1,y) &&
         imageValue(x,y) > threshold then

         if ttl[x] == 0 then
            finalSeeds.Add(x,y)
         end if

         for i = (x – seedRadius) to (x + seedRadius)
            ttl[i] = seedTTL
         end for

      end if
   end for

   for x = 0 to (imageWidth – 1)
      ttl[x] = max(0, ttl[x] – 1)
   end for
end for

end function
```

Figure 11: The seed extractor pseudo-code

In our application, the best balance between reliable positive detection and rejecting false positives was obtained by setting both `seedTTL` and `seedRadius` to 5 pixels. The TTL buffer values at all scanning lines when processing a sample input can be seen in Figure 14. Notice the algorithm rejecting a vast portion of the skull while recognizing most of the vessels.

4.2.5 Summary

To recapitulate the algorithm of our automated seed extraction, we extract the edges from the image at multiple scale ratios, apply a Hough transformation to find circular objects and extract seeds from the image. These seeds are then fed to the semi-automatic segmentation engine described in section 4.1. The seed generator, however, has no memory of seeds produced in its previous passes or of the vessel trees created by the semi-automatic segmentator. Therefore it may re-detect vessels that have already been processed.

To deal with this problem, we use a 3D search structure (in our case a uniform 3D grid) to store the locations of already segmented areas. Let us consider the vessel tree resulting from the segmentation. Its points could be considered small cylinders, whose axes lie on the vessel center line. We store them into our grid after every segmentation pass. So, when trying to extract seeds from the next slice, we check all the candidate seeds against that grid. If there is a collision, we assume that the vessel indicated by the candidate seed has already been processed and that seed is not passed to the segmentation stage. In Figure 14, these accepted and rejected seeds can be observed. The same seed culling procedure is applied on data from other scales using the same instance of the grid.

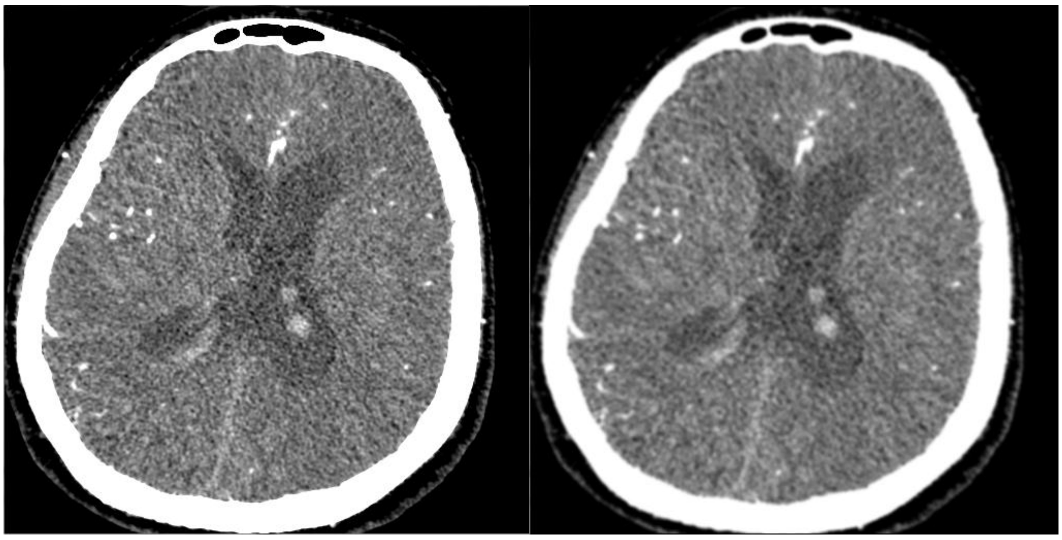


Figure 12: Left - original slice; Right - Gauss-filtered slice

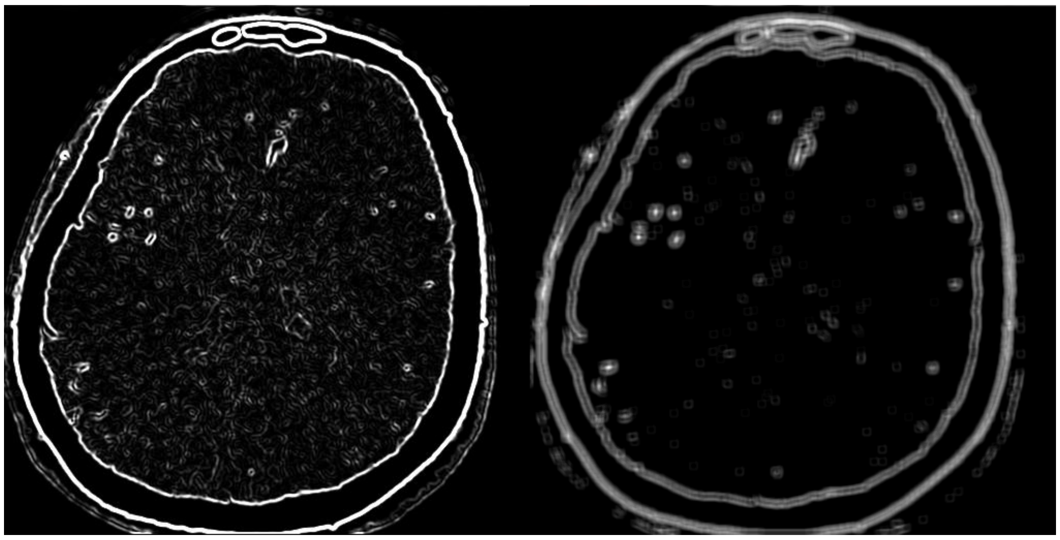


Figure 13: Left - response of the Sobel edge detection filter; Right - output of the Hough transform

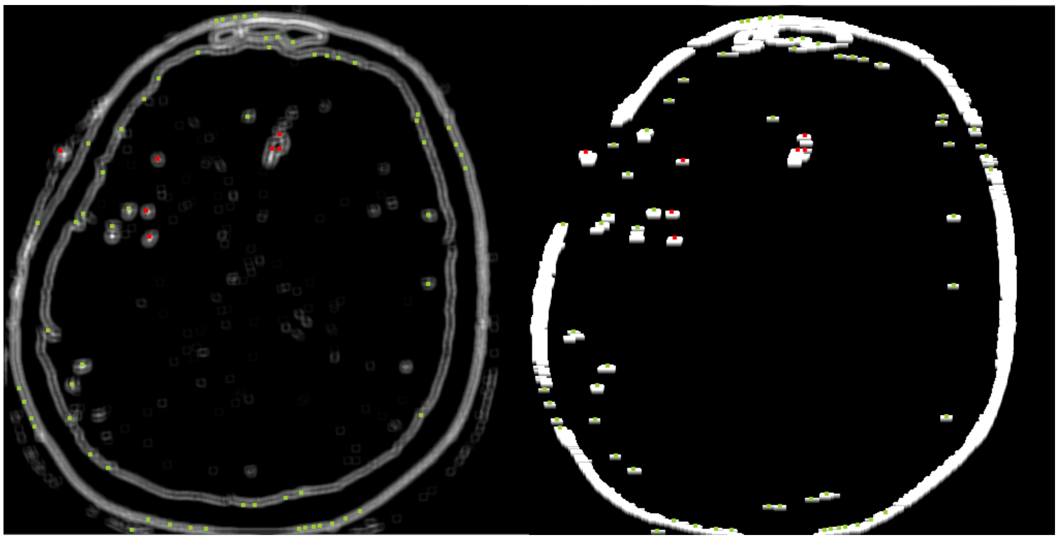


Figure 14: Left - output of Hough transform with recognized seed points (green - accepted, red - rejected); Right - TTL buffer of the seed extraction algorithm at each line with recognized seed points

4.3 Visualization

Now that we can segment a path or a tree of a vessel given a seed and also automatically generate such seeds, we need to properly visualize the user-selected part of that tree. We will use a modification of straightened CPR described in the previous chapter. Our version of CPR will, not unlike the other versions, ride the centerline of the vessel and generate cross-section snapshots of that vessel at regular intervals. This is the difference from straightened CPR that only extracts one line-of-interest from these vessel cross-sections. We then take these extracted cross-sections and stack them onto one another creating a new volumetric image that we will visualize by an appropriate method.

Furthermore, (19) was somewhat unclear as to calculating the $\mathbb{u}$ and $\mathbb{v}$ vectors that would define our cross-section plane. We will use the Bishop frame to calculate these so that our visualized vessel is subjected to as little twist as possible, which should further improve the visualization.

4.3.1 Curve-Local Coordinate System

In order to provide optimal $\mathbb{u}$ and $\mathbb{v}$ vectors for our CPR variation (Figure 15), we need to construct a local coordinate frame on every point on the vessel centerline as we traverse it. There are several known methods to construct a local coordinate frame on a space curve. These include the Frenet frame and the Bishop frame.

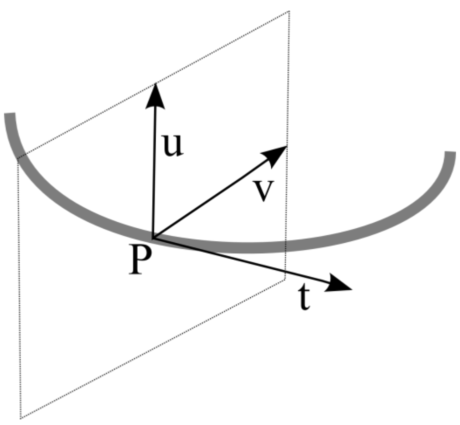


Figure 15: The local frame on the curve

Both the Frenet and Bishop frames generate the first derivative of the tangent $\mathbb{t}$, normal $\mathbb{u}$ and binormal $\mathbb{v}$ vectors using the parameters of the curve. The Frenet frame is defined (20) as:

$$\frac{d}{ds}\begin{pmatrix}\mathbb{t}\\ \mathbb{u}\\ \mathbb{v}\end{pmatrix} = \begin{pmatrix}0 & \kappa & 0\\ -\kappa & 0 & \tau\\ 0 & -\tau & 0\end{pmatrix}\begin{pmatrix}\mathbb{t}\\ \mathbb{u}\\ \mathbb{v}\end{pmatrix} \tag{12}$$

Where κ is the local curvature of the curve, otherwise defined (25) as the reciprocal of the osculating circle radius and τ is the curve torsion. Curve torsion (26), in the simplest of terms, can be understood as the amount by which the curve's osculating plane changes. The Frenet frame, however, imposes further restrictions on our space curve γ. It requires that $\gamma \in C^3$ and that γ be non-degenerate, in other words, the curve's derivatives γ' and γ'' be linearly independent.

The other approach we are considering is the Bishop frame, otherwise known as the parallel transport frame. It is defined by the equation:

$$\frac{d}{ds}\begin{pmatrix}\mathbb{t}\\ \mathbb{u}\\ \mathbb{v}\end{pmatrix} = \begin{pmatrix}0 & k_1 & k_2\\ -k_1 & 0 & 0\\ -k_2 & 0 & 0\end{pmatrix}\begin{pmatrix}\mathbb{t}\\ \mathbb{u}\\ \mathbb{v}\end{pmatrix} \tag{13}$$

Where k_1 and k_2 are coefficients for whom:

$$\kappa(t) = \sqrt{k_1^2 + k_2^2} \tag{14}$$

And

$$\tau(t) = \theta' \tag{15}$$

Where

$$\theta(t) = arctan\frac{k_2}{k_1} \tag{16}$$

Effectively, k_1 and k_2 are the Cartesian coordinate equivalents for the polar coordinates κ, θ. There is another feature of the Bishop frame that makes it more appealing to us, that is requires only C^2-continuity for our space curve and the non-zero first derivative of the curve, γ'.

We obviously need to calculate the k_1 and k_2 coefficients. We do this by simply projecting the vessel tangent vector of the next sampling onto the current $\mathbb{u}$ and $\mathbb{v}$, as is illustrated by:

$$\begin{gathered} k_1 = \mathbb{t}(i+1) \cdot \mathbb{u}(i) \\ k_2 = \mathbb{t}(i+1) \cdot \mathbb{v}(i) \end{gathered} \tag{17}$$

$\mathbb{t}, \mathbb{u}, \mathbb{v}(i)$ being the vessel tangent, normal and binormal vectors, respectively at the position i. And of course, all of these vectors are normalized.

In our implementation we may need to walk the curve with a greater resolution than the segmatation yielded. To achieve sufficiently smooth results, we calculate the cutting plane vectors at two successive centerline points P_0, P_1 and linearly interpolate the normal and binormal vectors.

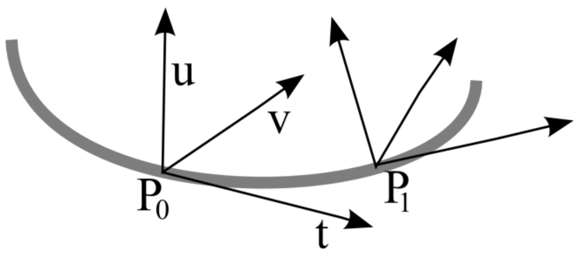


Figure 16: Set-up for interpolating the u and v vectors with sub-slice precision

So, in our implementation when starting the visualization, we know the seed point and the curve tangent at that point. As the frame transport provides only first derivatives of the $\mathbb{u}$ and $\mathbb{v}$ vectors, we need to calculate their initial values. We do this simply by determining which axis vector is at the greatest angle to the vessel tangent. This vector is then projected to the first vessel cutting plane and normalized to a unit-vector.

Then the other vector is calculated as the cross product of the tangent and our previously-calculated vector $\mathbb{v} = \mathbb{u} \times \mathbb{t}$. This makes it perpendicular to both vectors, which exactly fits the bill.

4.3.2 Extracting Cross-Section Images

At this point, we have the local coordinate system at each point of the curve defined. The next step would be to calculate the cross-section rectangle $ABCD$. We define the necessary points as

$$\begin{aligned} A &= P - r\mathbb{u} - r\mathbb{v} \\ B &= P - r\mathbb{u} + r\mathbb{v} \\ C &= P + r\mathbb{u} - r\mathbb{v} \\ D &= P + r\mathbb{u} + r\mathbb{v} \end{aligned} \tag{18}$$

Where r is the parameter that defines how large the sampled vessel cross-section will be. Now that the corner locations for the cross-section rectangle are known, we sample every point in that rectangle ($2r \cdot 2r$ points for the sake of completeness). Smooth sampling is accomplished by using trilinear interpolation.

4.3.3 Visualizing the Cross-Section

After the individual vessel cross-sections have been sampled and stacked one onto another, we need some way of visualizing this volume data. We implemented two methods, the maximum intensity projection (MIP) and a modification that calculates the average value of the ray-bounding box intersection path, which we will simply call averaging (AVG). Both are well known and simple enough to implement efficiently. Due to the proportions of the volume box we need to visualize, particularly its large width and small height and depth, we chose to render it without perspective. Therefore we use a perpendicular projection.

Maximum intensity projection is a simple method that basically shoots a ray from the camera through the visualized volume. If the volume is not intersected, the corresponding pixel is left black (or at the background color). A more interesting case occurs when the ray does hit the volume. A ray can be defined using the parametric expression:

$$X = A + t\mathbb{d} \tag{19}$$

Where A is the ray origin and $\mathbb{d}$ is its direction. Using an analytical expression of the volume's bounding box we will get the parameters, t_0 and t_1, of the places where the ray will enter and leave the volume bounding box as depicted in Figure 17.

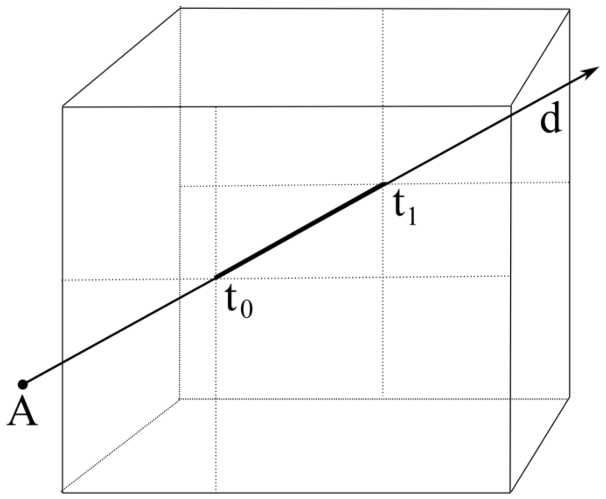


Figure 17: Ray intersecting a box

When these parameters are known, the MIP in that pixel can be expressed as:

$$MIP = \max\{v(X) \mid X = A + t\mathbb{d};\ t_0 \leq t \leq t_1\} \tag{20}$$

The quantity $v(X)$ is the value of voxel at the position X. In essence, as it name may hint, it is the maximum image value along the ray's path (27). The other projection, AVG does not differ much. It is calculated as the average value along the ray's path, instead of maximum. The equation then becomes:

$$AVG = \frac{1}{t_1 - t_0} \cdot \int_{t_0}^{t_1} v(A + t\mathbb{d}) \cdot dt \tag{21}$$

As we are dealing with voxel grid, the equations 20 and 21 can be discretized and simplified to nothing more than a maximum or an average of voxel values along the corresponding row or column of the straightened vessel image. This simplification proves useful in optimizing the implementation.

As the speed tests in the next chapter will show, this method is slower than normal straightened CPR as a result of extracting the whole vessel cross-section square, rather than just one line-of-interest. This speed degradation can be alleviated by a more coarse quantization of the t parameter in Equations 20 and 21. We have implemented such subsampling by reducing the number of depth samples in the cross-section extraction stage. This subsampling ratio is configurable and provides a good way to balance speed and rendering quality. The speed gains are measured in the next chapter.

5 Results

In this chapter we will examine the results given by our segmentation and visualization methods. We will test the stability and precision of the segmented path on two synthetic datasets with varying amount of added noise. Next, comparisons between straightened CPR (as shown in (19)) and our enhanced CPR will be given. Finally, the speed of seeded segmentation, automated segmentation and visualization will be compared on a mainstream desktop PC.

5.1 Results on Synthetic Data

We generated two sets of synthetic data to use for testing the methods proposed in this thesis. Both sets were volume images, 256x256x256 voxels large. The first one contained a straight vertical vessel with its background filled with varying levels of white noise. The other set contained a single helix, again, with varying amounts of noise. We tested the precision of the detected vessel path compared to the path our generator created. The second test concerned the visualization. Our synthetic data has been generated by a little utility we designed for this purpose. Further details are in Appendix C.

5.1.1 Testing Segmentation Stability

For the purposes of testing the stability and precision of our semi-automatic segmentation algorithm we generated two sets of synthetic data. This test examined how precise the segmented path is with respect to the level of additive white noise. We compared the segmented path node by node to the intended path (used by the generator). The Euclidean distance between the intended and detected positions was our error metric.

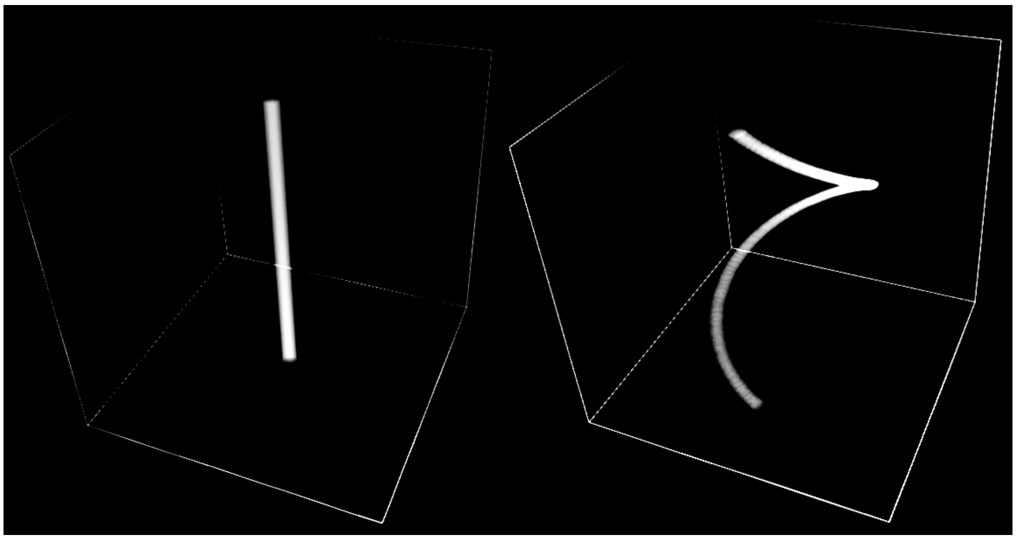


Figure 18: Synthetic testing data, both with no noise added: Left - straight tube; Right - single helix

The noise in our test images is a spectrally unfiltered noise sampled from a random number generator (we assume the generator's output values are uniformly distributed) and scaled to the desired amplitude.

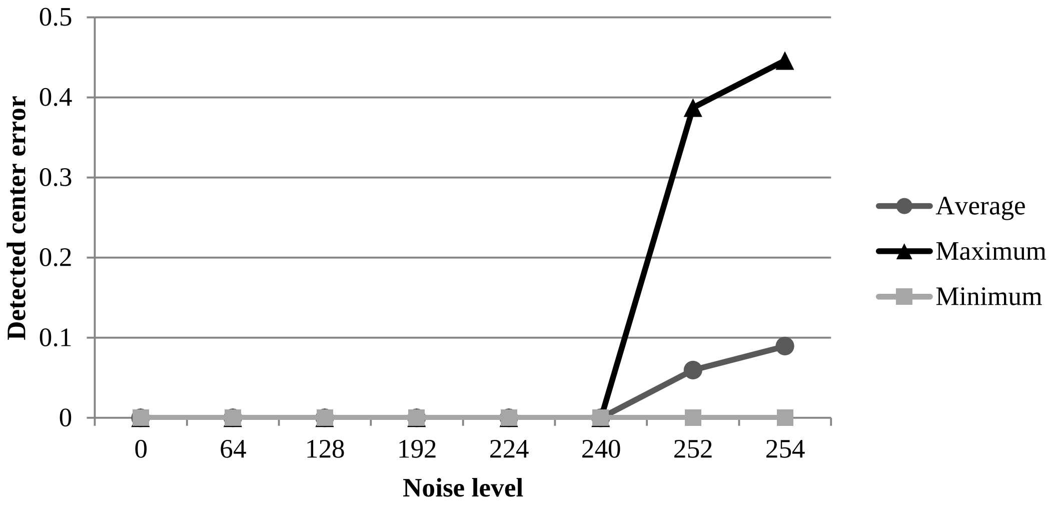


Figure 19: Segmentation error in detecting the centerline of the straight line data

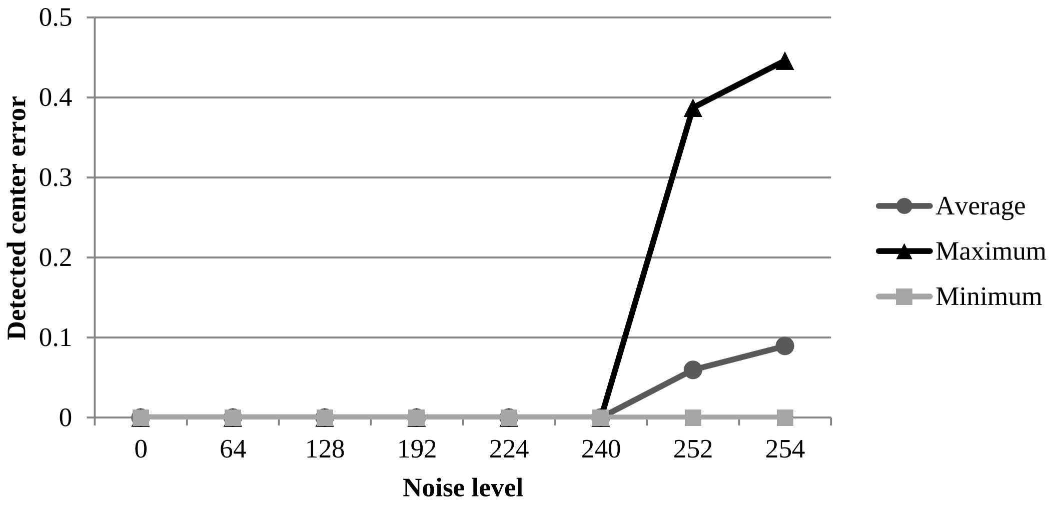


Figure 20: Segmentation error in detecting the centerline of the helix data

As indicated by the results in Figures 19 and 20, the semi-automatic segmentation copes well with noise in the synthetic datasets as long as the noise stays out of the region growing algorithm's tolerance. However, even with the white noise strong enough to reach well into the threshold interval of the region growing (noise levels 252 and 254), the vessel centerline is misdetected only by less than half of a voxel.

5.1.2 Visualization Test

The visualization test on synthetic data consisted of running the visualization on a part of the segmented synthetic vessel and looking for any visible artifacts. Figure 22 shows the same segment of an artifical vessel with three different noise levels traced and visualized. Note the vessel in the bottom part of Figure 22 is barely distinguishable by the human eye, however the segmentation engine managed to calculate its path with very little error. Figure 21 shows that same vessel segment in 3D space.

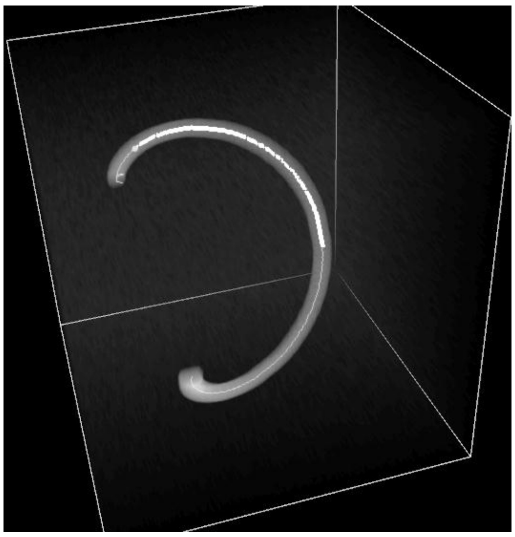
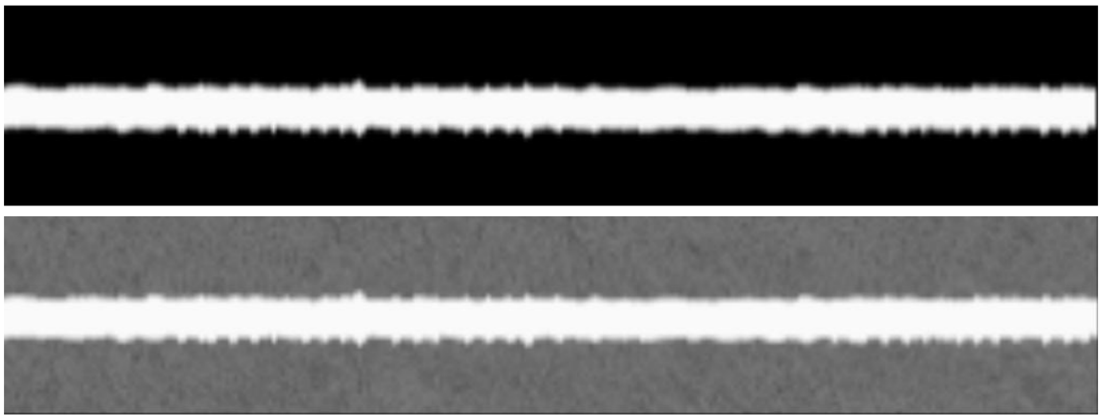


Figure 21: The helix, its visualized part is highlighted with a thick line along its center

There are also no serious visible artifacts from the vessel path sampling, path miscalculation or local curve frame discontinuity. The jagged edges visible in noiseless image are caused by the fact that the synthetic vessel data itself was not antialiased. Thus, the visualization appears to be performing within the expected parameters.



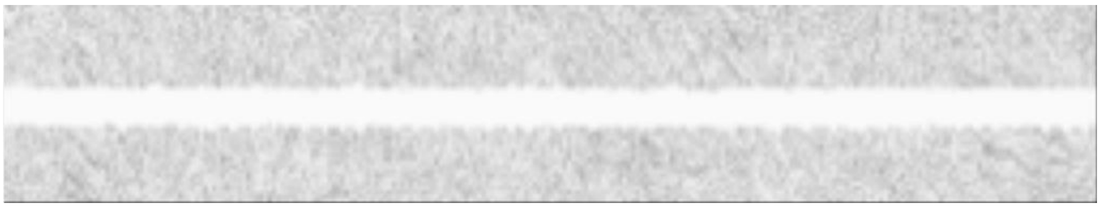


Figure 22: A visualized segment of a vessel (MIP projection), Top - no noise, Center - noise level at 128, Bottom - noise level at 252

As stated in previous chapters, spatial orientation can be somewhat difficult with straightened CPR (19). Because our MIP-CPR and straightened CPR are closely related, it would stand to reason that our visualization method inherit the same problem. A closer inspection of the results, however, shows these effects somewhat mitigated. Figure 23 contains two renderings of the same vessel segment with two different methods configured with the same values.

The comparison clearly shows the straightened CPR with considerably less clutter. The line of interest, obviously, did not intersect any other significant features. In contrast to that, our method extracts a whole rectangle-of-interest capturing more of the vessel's surroundings which are visible in the MIP rendering. Vessel bifurcations are clearly visible, giving more clues to the vessel's spatial orientation. However, only experience may show if the increased computational demands of this method are sufficiently justified by these visual enhancements.

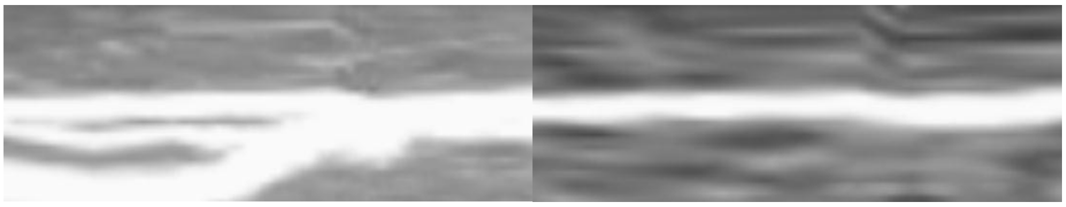


Figure 23: A blood vessel segment Left - visualized with MIP-CPR, Right - visualized with conventional straightened CPR

5.2 Results on Real Data

As our real-life datasets, we used a CT scan of a human head with a contrast agent applied. This is a rather complex dataset with vessels ranging from moderately large to small, running perpendicular to the slices or almost parallel. The data contains some noise and many of the vessels are embedded in the brain or running close to the cranium. There is also considerable branching and curvature of the vessels.

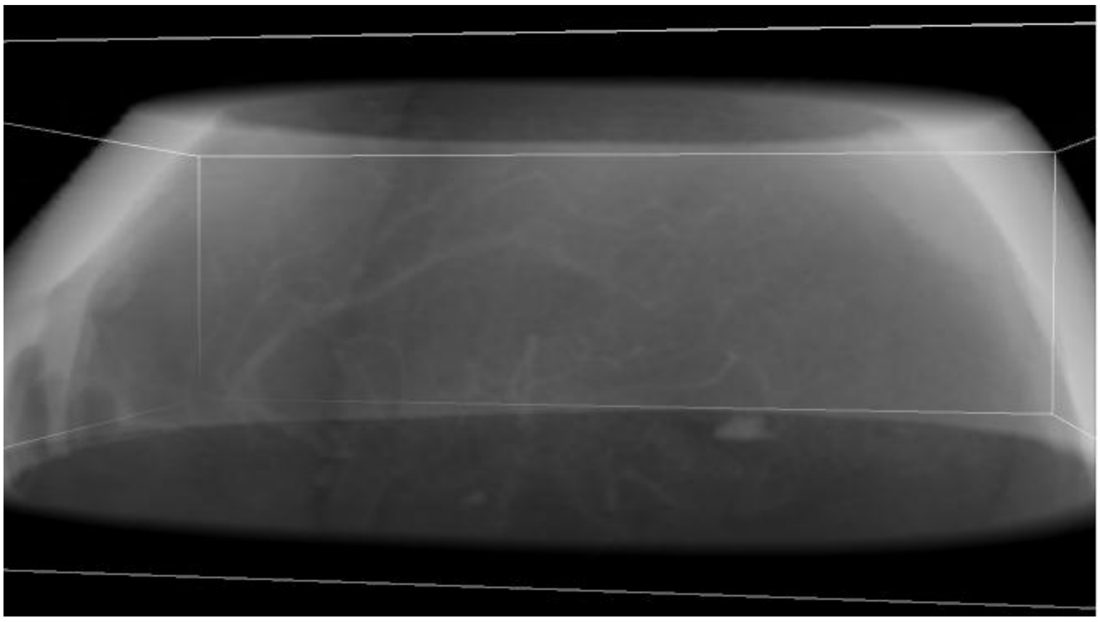


Figure 24: The section of human head that we used as the real-world data, *angio1*

We used the dataset in Figure 23 for testing the automatic segmentation.

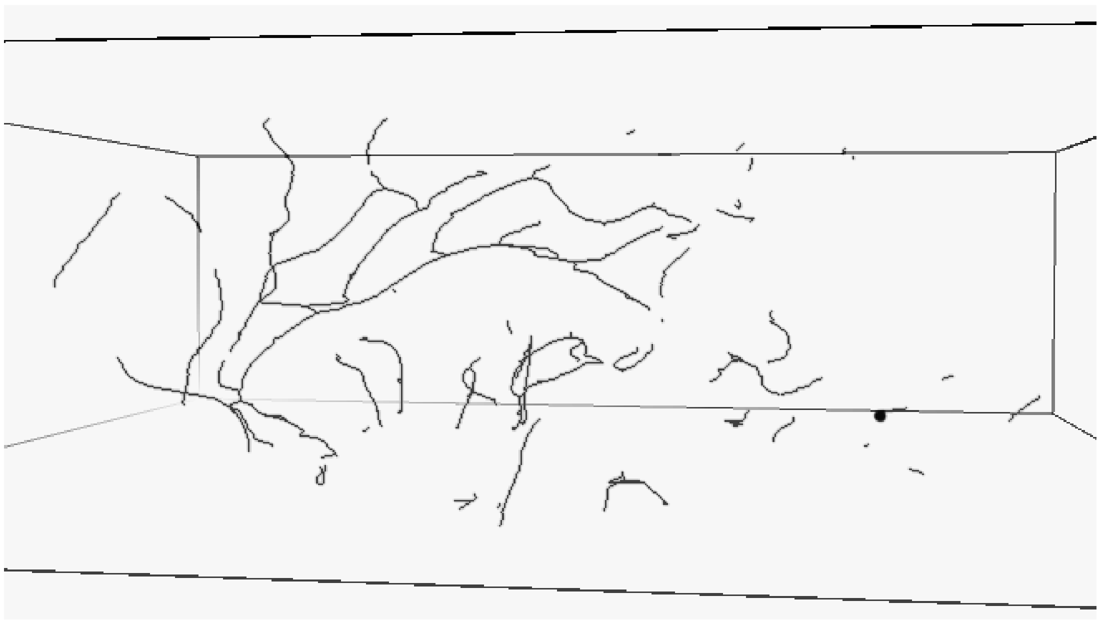


Figure 25: The vessel tree detected by our segmentation methods in the human head cross section (inverted colors)

5.3 Speed Tests

We ran our speed tests on a desktop computer. It features an Intel Core i7-920 processor with 3x 1GB DDR3-1066 and runs an x86-64 version of Microsoft Windows 7. Our speed tests included semi-automatic segmentation of short vessels, automated segmentation of two models and visualization in three different modes.

5.3.1 Semi-Automatic Segmentation

For this test, we used a simple setup. On the loaded model *angio1* we manually induced segmentation on several spots that contained vessel cross-sections. We then measured the time it took to finish segmentation of that vessel subtree. We also recorded the count of nodes produced by the segmentation procedure so that we can calculate time per one node.

Our test sample included chains of 32 to 77 nodes. The shortest took 11 milliseconds to segment, producing a speed of 1727 nodes per second, further referred to as nps. The longest chain took 40 ms, resulting in a speed of 1925 nps. These faster processing speeds with longer chains are probably the result of one-shot memory management overhead amortized over a longer real processing time.

5.3.2 Automatic Segmentation

In this test, we examined the processing speed of automated segmentation. The test consisted of measuring the time taken by the full automatic segmentation of both *angio1* and *angio2*. We also counted the total number of vessel nodes produced in the segmentation. The results are given in Table 2.

	nodes	time	speed
angio1	1489	4.78 s	311.5 nps
angio2	4072	15.42 s	264 nps

Table 2: Fully automatic segmentation speed results

As the results indicate, the vessel node generation speeds appear almost an order of magnitude slower than those of semi-automatic segmentation even with short strands. It must, however, be kept in mind that the automatic segmentation does considerable image processing and seed extraction before the semi-automatic segmentation part can be executed. Using the estimates of semi-automatic node generation speed and the automatic speed, we can estimate how much time is consumed with preprocessing and how long the actual segmentation takes. For this estimation we will assume a processing speed in the middle of the interval produced by testing in the previous section, 1825 nps.

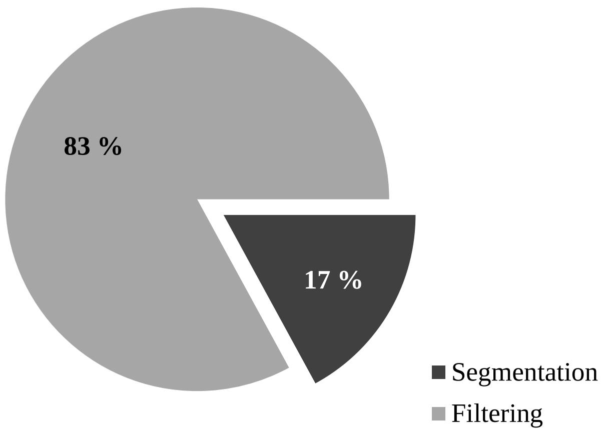


Figure 26: The estimated fractions of time spent on segmentation and image filtering

For the smaller image, *angio1* we estimated that segmentation ran only 0.82 seconds out of the total 4.78 seconds. This indicates filtering taking the vast majority of processing time, almost 83%. This difference is even more pronounced in the other sample, *angio2*. The total filtering time reached above 85.5%. This could be explained by the fact that the farther the automatic seed generation proceeds, the more of the vasculature has already been processed. This results in higher seed rejection ratio, however the same amount of time must be taken to generate these seeds in the first place. Consequently, more time is spent doing filtering.

In any event, the filtering is a bottleneck in this case. Fortunately, seed generation can be made to run parallel as the individual slices on which the filtering is done are in no way interdependent. Further acceleration can be achieved by running filtering and seed extraction on GPGPU, whose massive parallelism is well suited for such tasks. These enhancements will be left for future improvements.

5.3.3 Visualization

This test examines the speed of our visualization methods. Again, we measured the time necessary to visualize a vessel segment given the appropriate vessel tree and a starting node. We compared the two methods proposed in this thesis, MIP-CPR and AVG-CPR. To put these results in context, we also implemented and tested standard straightened CPR as introduced by (19). The test itself was run on all three methods with the same vessel segment and other parameters (e.g. line-of-interest size, tracking step, initial normal rotation).

The tests were run on all three methods with equal parameters. The square-of-interest was 48x48 pixels large; in the case of straightened CPR the line-of-interest was set to 48 pixels length. The initial rotation of the Bishop frame was set to 0 degrees in all three cases. The vessel was tracked in length increments of 0.1. In total, 414 slices of the vessel were extracted by each method. The Table 3 shows the average times taken to visualize the common vessel segment.

	MIP-CPR	AVG-CPR	Straightened CPR
Average time	330 ms	340 ms	19 ms
Slices per second	1254	1217	21783

Table 3: Visualization speed test results

The results show the Straightened CPR is more than an order of magnitude faster than our new methods, MIP-CPR and AVG-CPR. This is, however, not surprising considering the fact that these methods sample the volume data in $M^2 \cdot N$ points as opposed to the straightened CPR's $M \cdot N$ points, where N is visualized path length and M is the square-of-interest edge size or the line-of-interest size, respectively. These processing times are still not exceedingly long and these methods can be safely called interactive. The speed issue is further mitigated by the fact that our enhanced CPR produces images with more spatial context, simplifying orientation.

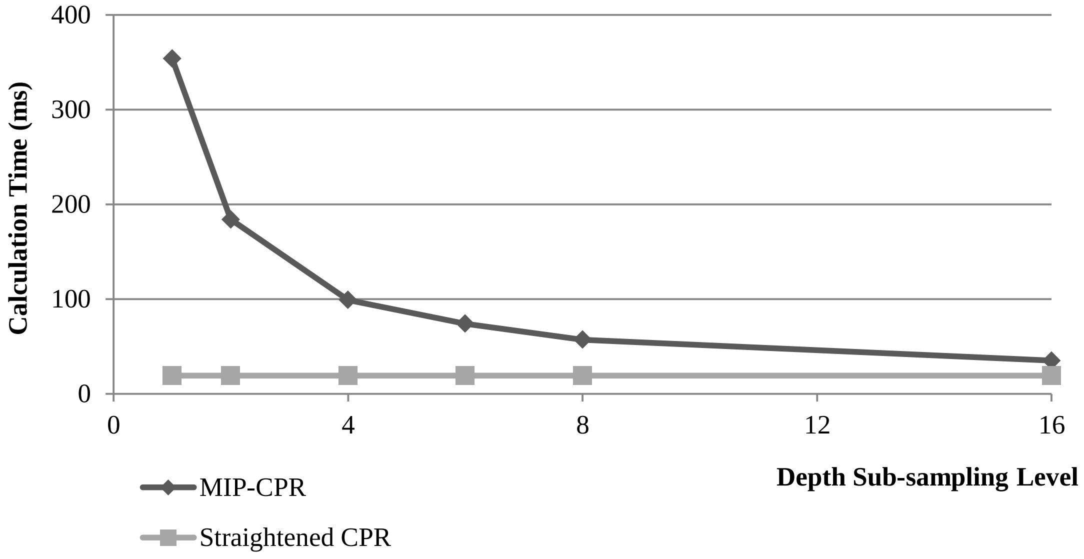


Figure 27: MIP-CPR calculation time depending on depth subsampling level

Furthermore, the speed and visualization quality can be balanced by reducing the number of depth samples for the projection. To enable such balancing, we implemented such subsampling and tested in the same conditions as the previous part of this test. Our results show that even 6x or 8x subsampling produce images of acceptable visual quality and come much closer to straightened CPR execution time. The same vessel segment took 19 milliseconds to visualize with straightened CPR and 57 ms with 8x subsampled MIP-CPR. The image rendered with subsampled MIP-CPR kept most of the features of the fully-sampled one, yet took only about one seventh of the time to calculate. Albeit still several times slower than straightened CPR, the method is obviously fast enough, potentially producing about 18 frames per second of visualized data on our testing computer. Further speed improvements can be made by implementing the visualization in parallel or on a GPGPU.

Conclusion

This thesis provides an overview of the segmentation and visualization methods used in medical applications, especially in angiography. We have also designed and tested a system for blood vessel segmentation and visualization on CT data with contrast agent applied. We divided that system into three distinct parts: semi-automatic vessel segmentation, automated seed generation and visualization.

We used a region growing algorithm to determine the vessel cross section in a particular volume data slice. Several enhancements and heuristics were used to improve the performance of this method, most notably an over-segmentation prevention heuristic and a hole-filling algorithm to improve the algorithm's stability on noisy data. The results of segmentation on individual slices are converted on-the-fly into a vessel graph, allowing for branching. A centerline detection algorithm based on center of mass calculation has proven sufficiently precise on both synthetic and real-life data.

The automatic seed generation part of our system consists mainly of image filtering. First, we extracted the edges with a Sobel filter preceded by a Gaussian blur. Exploiting the fact that blood vessels will mostly appear as small circular objects (unless their path is almost parallel to the slice plane) we used a Hough transform for circles to find the blood vessel candidates. This is done in multiple scales to enable detection of vessels of different sizes. We then designed a simple algorithm for extracting seeds from images processed with the above-mentioned chain of transformations that immediately rejects false positives created mostly by large objects that are not vessels and generates but one seed for a detected vessel. This list of vessel candidates or seeds is fed to the semi-automatic segmentation engine. To avoid running the segmentation repeatedly on already processed vessels, and consequently improve processing speed, we used a uniform grid structure for storage and fast detection of previously segmented areas. After the automated segmentation is done, a fast sweep eliminates very short disjoint vessel paths, which were probably falsely detected and attempts to join the existing vessel paths into larger structures.

For visualization, we modified the well-known straightened CPR to display not only a single line-of-interest, but to construct a vessel projection from its whole cross-section. The method has proven slower than original straightened CPR; however we believe that while very small vessel calcifications, especially on larger vessels may not be reliably captured by straightened CPR, our method should show them. By capturing the whole surroundings of the vessel, our method provides potentially better spatial orientation based on the output image. For balancing the visualization speed and quality, we use depth subsampling that reduces the number of columns calculated in the cross-section sampling stage and significantly improves processing time. We also used a Bishop frame to slide the cutting plane along the vessel’s path in a manner that would produce minimal twist to further improve the visualization.

Despite the encouraging results on synthetic data and our real-life data, there is still room for improvements in the areas of processing speed and possibly noise resilience. To that end, automatic seed generation can be made to run in parallel by processing several slices simultaneously. Noise resilience may be improved by modifying our region-growing algorithm with additional heuristics or replacing it entirely. Semi-automatic segmentation can be run in parallel as well, the only difficulty being synchronization of the grid structure that prevents re-segmentation of already processed vessel segments. Our visualization method, MIP-CPR is sufficiently fast even with little Z-subsampling, albeit slower than conventional stratightened CPR. If required, speed improvements could be gained by running the cross-section extraction in parallel or on a GPGPU.

A Programmer's Guide

The `VesselVisualizer` solution consists of two projects. The first one, `VesselVisualizer` is the project that implements all the segmentation and visualization techniques we designed in Chapter 4 and the UI for displaying the results. The second project is the VL library designed and implemented by Hlaváček in (28) and modified to use the OpenTK library instead of Tao for interfacing OpenGL in .NET environment. Therefore we will discuss only the first project, `VesselVisualizer`.

In order to add new segmentation or visualization methods, the programmer should be aware of two interfaces – `IVesselVisualization` and `IVesselSegmentation`. Complete programmer's reference can be found on the attached CD.

A.1 `IVesselSegmentation` interface

This interface must be implemented by all semi-automatic methods. Its methods provide for seeded segmentation of a vessel in a volume data slice, extracting the seed from a previously segmented mask and for extracting the vessel subtree indicated by a seed.

`bool SegmentMask(IVolume vol, Point seed, int slice, ref BitMask2D mask);`
This method starts segmentation at the position indicated by arguments `seed` and `slice` in the volume data `vol`. The segmented parts are returned in the `mask` output argument by setting the cells that correspond to vessel location to `MASK_TRUE`. Method returns `true` if segmentation succeeded and `false` if segmentation failed for any reason.

`bool ExtractSeed(BitMask2D mask, out PointF seed);`
The method extracts the seed for the segmented vessel defined by the `mask` argument and returns it in the `seed` output parameter. `true` is returned if the seed has been extracted and `false` if the extraction failed for any reason (like empty mask).

`bool CreateVesselTree(IVolume vol, Point seed, int seedSlice, out VesselTreeNode root);`
This method is called by the framework to extract maximum possible portion of the vessel graph starting at the `seed` point on the slice `seedSlice`. The vessel subgraph is returned in the `root` argument. The method is to return `true` if the operation succeeded and `false` otherwise.

A.2 `IVesselVisualization` interface

The `IVvesselVisualization` interface must be implemented by all vessel visualization algorithms. Its methods provide for configuring input and output parameters and for two-stage rendering. The rendering part is deliberately split into preprocessing and rendering itself. This enables greater efficiency in dealing with repeated visualization of the same vessel segment but with different parameters (like viewing angle).

`bool SetVolume(IVolume vol);`
This method sets the volume data to be further used by the visualizer. The call ends with `true` if the volume data is valid and with `false` if the data cannot be used or an unexpected error has occured.

`bool SetOutputParams(int sliceCount, int crossSectionSize, float crossSectionSpacing, float rotation);`
The `SetOutputParams` method configures the visualization. Upon the next rendering, the visualizer will attempt to render `sliceCount` vessel cross sections, `crossSectionSize` pixels large. The vessel will be cut and sampled in `crossSectionSpacing` voxels increment. The `rotation` argument sets the initial roation of the transport frame along the vessel path, in degrees. A successful call returns `true`, while `false` is returned when the parameters are invalid.

`bool Preprocess(VesselTreeNode start, out List<Point3F> path);`
Preprocessing is initialized with this method. The parameter `start` indicates at which vessel graph node to start visualizaion, the output argument `path` will hold the list of points along the vessel center line traversed when visualizing the vessel. The method returns `true` if the preprocessing succeeded and `Render` can be called, `false` otherwise.

`bool Render(Bitmap bmp);`
This method converts the preprocessed data into a `System.Drawing.Bitmap` object `bmp` that can be instantly dipslayed. When all succeeds, `true` is returned, `false` indicates failure and in that case the bitmap will contain invalid data. Please note that the `bmp` argument contains an already created and initialized bitmap.

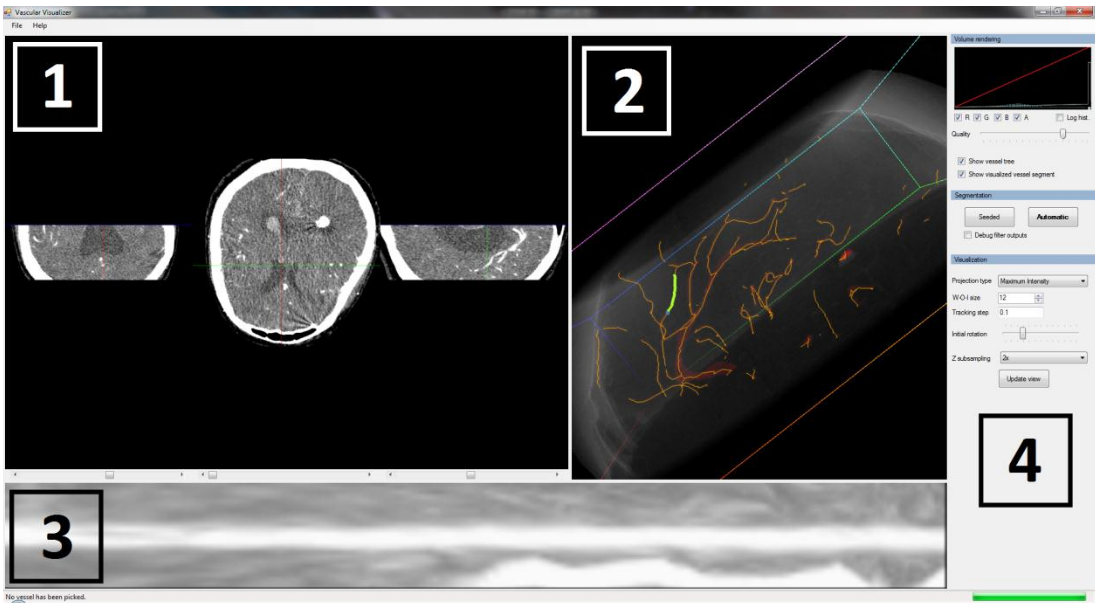
B User Documentation

The user interface (UI) of the application has been designed to provide fast and intuitive access to its functions. In this appendix we will give a brief overview of the main window and the functions it provides.

B.1 User Interface

The main window consists of four main elements, as shown:

1. Volume cross-sections view
2. Direct volume rendering of the volume data, with segmented mask and vessel tree
3. Vessel visualization window
4. Configuration side-bar



Show vessel tree [✓]
Show visualized vessel segment [✓]
Debug filter outputs [ ]
Projection type: Maximum Intensity
W-O-I size: 12
Tracking step: 0.1
Initial rotation
Z subsampling: 2x

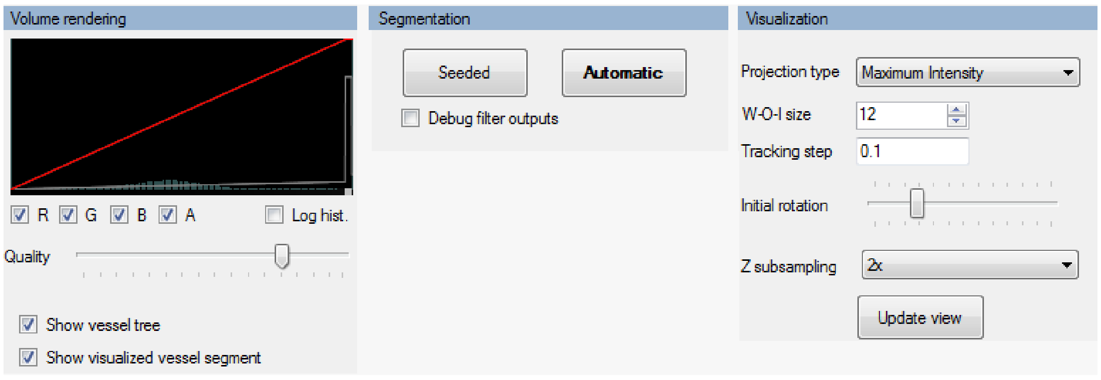
No vessel has been picked.

Figure 28: The main window of VesselVisualizer

Now, let us briefly list the functions of the main window's respective parts. Part 1 gives a view of three cross-sections of the loaded volume: coronal, saggital and transversal. The planes defining the cross-section can be shifted along their normals by the scroll bars in the bottom part of this window element. The view of all three cross-sections automatically updates as any of the scroll bars is shifted. Furthermore, this window is used for seeding the semi-automatic segmentation. After the user clicks on the desired point in any of the three cross-section visualizations a point appears both in 2D and 3D visualization to simplify ascertaining its position. When the user is satisfied with the seed's position, clicking the *Seeded* button in element 4 starts the segmentation.

Element 2 is the direct volume rendering of the loaded volume data. A file can be loaded by using the *File>Open…* menu item. Supported files include RAW volume files and DICOM. Processing is, however, enabled only on 8-bit RAW files. The volume rendering transfer function can be adjusted in the side-bar. Model can be rotated by holding down the middle mouse button and dragging. The field of view and incidentally zoom can be altered with the mouse wheel. This window also visualizes the mask of the segmented vessels in red and the vessel tree if configured to do so in the side-bar. Clicking a vessel in this window causes the visualization to be calculated starting at the selected point. Then the visualization is displayed in element 3. When this is done, the visualized segment is highlighted with a thick line in the volume rendering to ease the orientation in both images.

The configuration side-bar provides controls for three separate parts of the program: *volume visualization*, *segmentation* and *vessel visualization*. Let us go through them one by one.



☑ Show vessel tree
☑ Show visualized vessel segment
☐ Debug filter outputs
Projection type: Maximum Intensity
W-O-I size: 12
Tracking step: 0.1
Initial rotation
Z subsampling: 2x

Figure 29: The configuration side-bar

The volume rendering part allows for volume rendering transfer function modification even for individual color channels. The *Quality* slider adjusts the number of samples taken when sampling the volume data per ray. A higher value gives better rendering quality, but the rendering performance may deteriorate. Next, the checkboxes set whether or not to visualize the vessel tree produced by the segmentation and the specific part of it used for vessel visualization.

The next part is rather Spartan, this was however the intent. The *Seeded* button starts the semi-automatic segmentation given a seed in the 2D cross-section view. The *Automatic* button starts the automatic segmentation whose progress is shown in the status bar and intermediate segmentation results are shown in the 3D volume view as it proceeds. The checkbox enables showing debugging outputs for the automated segmentation. After each pass, a window is shown containing the responses of all the filters used and the positions of the generated seeds.

The final part of the side-bar facilitates the configuration of our visualization methods. From top to bottom, *Projection type* selects the vessel visualization mode. The following modes are supported:

- Maximum Intensity (MIP-CPR)
- Averaging (AVG-CPR)
- Straightened CPR

W-O-I size sets the visualization radius. The line-of-interest will be twice the length of W-O-I size and the square-of-interest for our methods will be twice the W-O-I size in both height and width. *Tracking step* sets in how large increments the vessel will be tracked. *Initial rotation* specifies the angle by which the vessel normal will be rotated around its tangent the first time it is calculated. Basically it rotates the line-of-interest around the tangent for straightened CPR and sets the camera's position for rendering the generated volume of the vessel in the other two methods. The control's range is full 360 degrees. *Z subsampling* sets by what ratio the depth of the generated vessel volume will be subsampled. Setting higher values increases calculation speed but reduces the rendering quality. Finally, the *Update view* button applies the configuration changes and re-visualizes the previously selected vessel segment.

B.2 Hardware and Software Requirements

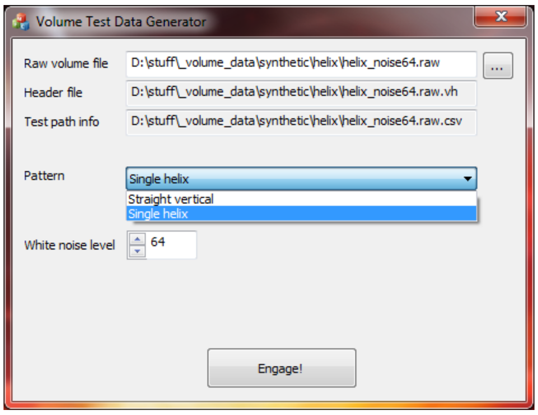
The application was built with the Microsoft .NET 3.5 framework and thus requires these files to run. An installation package[2] is included on the thesis CD.

The hardware requirements are dictated mostly by the VL library. A graphics adapter with Shader Model 3 or higher is required. This includes NVidia GeForce 6xxx series or later and ATI X1xxx series or later (28). Testing has shown that the program runs on Intel HD Graphics as well, however this is not recommended due to maximum volume texture size limitations[3] of that graphics adapter.

[2] Downloadable from https://www.microsoft.com/download/en/details.aspx?id=21
[3] Maximum 3D texture size limited to 256x256x256, insufficient for CT data

C Synthetic Data Generator Utility

In this thesis we relied in part on synthetic volume data, specifically the straight vertical vessel and a single-round helix with varying amounts of noise. To reliably generate such data we implemented a simple utility. Two patterns both of which we used in this thesis can be generated. Additive white noise level can also be configured. The size of the volume is hard-coded to 256x256x256 to allow for testing on less capable graphics hardware.


Volume Test Data Generator

Raw volume file: D:\stuff_volume_data\synthetic\helix\helix_noise64.raw

Header file: D:\stuff_volume_data\synthetic\helix\helix_noise64.raw.vh

Test path info: D:\stuff_volume_data\synthetic\helix\helix_noise64.raw.csv

Pattern: Single helix

White noise level: 64

Figure 30: The main window of testing data generator utility

Three files are generated in one session:

- The raw volume data itself
- A header describing the size and bit depth of the data conforming to what our testing program uses
- A comma-separated-values sheet containing the point coordinates along the centerline of the artificially generated vessel for segmentation path stability testing

This utility is written in Microsoft Visual C++ 2008 SP1 and thus requires the appropriate runtime libraries[4]. An installation package for these libraries, the program itself and its source codes are, of course, included on the attached CD.

[4] Installation package also downloadable from https://www.microsoft.com/download/en/details.aspx?id=5582

Bibliography

1. **Hsieh, Jiang.** *Computed Tomography: Principles, Design, Artifacts and Recent Advances.* s.l. : SPIE Press, 2003. ISBN 0-8194-4425-1.

2. **Doubek, Miloš.** Sledování arteriálního řečiště na CT datech. s.l. : Master thesis, MFF UK, 2008.

3. **Mathias Prokop, Michael Galanski, Cornelia Schaefer-Prokop.** *Spiral and multislice computed tomography of the body.* Stuttgart : Georg Thieme Verlag, 2003. ISBN 3-13-116481-6.

4. Introduction to CT physics. [Online] [Cited: July 24, 2011.] http://web.archive.org/web/20070926231241/http://www.intl.elsevierhealth.com/e-books/pdf/940.pdf.

5. Hounsfield scale - Wikipedia. [Online] [Cited: July 24, 2011.] http://en.wikipedia.org/wiki/Hounsfield_scale.

6. Contrast Medium - Wikipedia. [Online] [Cited: July 24, 2011.] http://en.wikipedia.org/wiki/Contrast_agent.

7. **Cemil Kirbas, Francis Quek.** A Review of Vessel Extraction Techniques and Algorithms. [Online] January 2003. http://www.ee.siue.edu/~sumbaug/RetinalProjectPapers/Review%20of%20Blood%20Vessel%20Extraction%20Techniques%20and%20Algorithms.pdf.

8. **A. Sarwal, A.P. Dhawan.** 3-d reconstruction of coronary arteries. *IEEE Conf. Eng. in Medicine and Biology.* 1994, Vol. 1.

9. **E. Sorantin, C. Halmai, B. Erdohelyi, K. Palagyi, L. Nyul, K. Olle, B. Geiger, F. Lindbichler, G. Friedrich, K. Kiesler.** Spiral-CT-based assessment of tracheal stenoses using 3-D-skeletonization. *IEEE Trans. on Medical Imaging.* 2002, Vol. 21.

10. *A locally adaptive region growing algorithm for vascular segmentation.* **Jaeyoun Yi, Jong Beom Ra.** 13, June 2003, International Journal of Imaging Systems and Technology, pp. 208-214.

11. **Falcao, A. X.** *Paradigmas de Segmentacao de Imagens Guiada pelo Usuario: Live Wire, Live-Lane e 3D-Live Wire.* s.l. : University of Campinas - UNICAMP, PhD thesis, 1997.

12. **A. X. Falcao, J. K. Udupa, F. K. Miyazawa.** An ultra-fast user-steered image segmentation paradigm: Live wire on the fly. *IEEE Transaction on Medical Imaging.* January 2000.

13. **Bartoli, A. Vilanova i.** *Interactive segmentation of medical images based on intelligent scissors.* s.l. : TU Wien, 1997.

14. **M. Couprie, Gilles Bertrand.** Topological Greyscale Watershed Transformation. *SPIE Vision Geometry VI Proceedings.* 1997, Vol. 3168.

15. **Krajíček, Václav.** *Měření objemu v 3D datech.* s.l. : MFF UK, Master thesis, 2007.

16. **D. Geiger, A. Gupta, L. A. Costa, J. Vlontzos.** Dynamic programming for detecting, tracking and matching deformable contours. *PAMI.* 1995, Vol. 17.

17. **Andrew Fitzgibbon, Robert Fisher.** A Buyer's Guide ro Conic Fitting. [Online] 1995. http://www.bmva.org/bmvc/1995/bmvc-95-050.pdf.
18. **Radim Halíř, Jan Flusser.** Numerically Stable Direct Least Squares Fitting of Ellipses. [Online] 1998. http://autotrace.sourceforge.net/WSCG98.pdf.
19. **Armin Kanitsar, Dominik Fleischmann, Rainer Wegenkittl, Petr Felkel, Meister Eduard Gröller.** CPR - Curved Planar Reformation. *VIS 2002, IEEE Volume.* 2002, Vol. 1, 1.
20. **McCreary, Paul Robert.** *Visualizing Riemann Surfaces, Teichmueller Spaces, and Transformation Groups in Hyperbolic Manifolds Using Real-Time Interactive Computer Animator (RTICA) Graphics.* Urbana-Champaign : University of Illinois, PhD thesis, 1998.
21. **Bourne, R.** *Fundamentals of Digital Imaging in Medicine.* s.l. : Springer, 2009. ISBN 9781848820869.
22. **M.S., Nixon.** *Feature extraction and image processing.* s.l. : Aguado A.S., 2008. ISBN 9780123725387.
23. **Leandro A. F. Fernandes, Manuel M. Oliveira.** *Real-time line detection through an improved Hough transform voting scheme.* s.l. : ScienceDirect, 2006.
24. **Dimitrios Ioannou, Walter Huda, Andrew F. Laine.** *Circle recognition through a 2D Hough Transform and radius histogramming.* s.l. : Image and vision computing, 1997.
25. **Weisstein, Eric W.** Curvature. *MathWorld--A Wolfram Web Resource.* [Online] [Cited: August 3, 2011.] http://mathworld.wolfram.com/Curvature.html.
26. —. Torsion. *MathWorld--A Wolfram Web Resource.* [Online] [Cited: August 3, 2011.] http://mathworld.wolfram.com/Torsion.html.
27. **Elliot K. Fishman, Derek R. Ney, David G. Heath, et al.** Volume rendering versus maximum intensity projection in CT angiography: What works best, when, and why. *RADIOGRAPHICS.* 2006, Vol. 26, 3.
28. **Hlaváček, Jakub.** *Zpracování medicínských dat na GPU.* s.l. : MFF UK, Master thesis, 2008.