NTNU
Norwegian University of
Science and Technology

Segmentation and Centerline Extraction of the Coronary Arteries with GPU Processing

Magnus Barlund Lefdal

Master of Science in Informatics
Submission date: July 2016
Supervisor: Frank Lindseth, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Coronary artery disease is one of the most common causes of death in Norway, and the world. While diagnosing this disease, and treating it is a common procedure at many hospitals, discovering and documenting a faster, safer and cheaper way to detect the disease could be critical for many patients. Our goal is to implement an algorithm that can automatically detect and produce a segmentation of the coronary arteries from Computed Tomography Angiography (CTA) images.

In this thesis we explore and discuss several different methods and approaches for segmentation of the coronary arteries and vessels in general in relation to compatibility for parallel execution on a GPU, accuracy and practicality. A suitable method is chosen based on the discussion and implemented.

The results show that the algorithm is able to segment large portions of the coronary arteries. But the produced results are vulnerable to noise, artifacts and irregular vessels. The most significant improvement would be a centerline selection addition, where the centerlines that are chosen to produce the segmentation is chosen based on their location in relation to the heart. Or some other method that is more reliable then the current bio-mechanical method. Our implementation is able to produce a segmentation and a centerline in around 10 minutes.

The method implemented in this thesis is a general approach to vessel segmentation, but tuned for coronary artery segmentation. The approach could potentially be adapted and used for extraction of other tubular structures.

Sammendrag

Koronarsykdom er en av de vanligste dødsårsakene i Norge, og i verden. Diagnosering og behandling av denne sykdomsgruppen er en standardisert og vanlig prosedyre på mange sykehus. Men siden det er stort behov for effektiv diagnostisering, og behandling så kan utviklingen og dokumenteringen av raskere, tryggere og billigere måter å oppdage koronarsykdom kan være svært nyttig for både pasienter og sykehus. Målet i denne oppgaven er og implementere en algoritme som automatisk kan oppdage og segmentere ut de koronararteriene fra et CTA bilde.

I denne oppgaven vil vi utforske og diskutere en rekke forskjellige metoder og tilnærminger til segmentering av koronararteriene og årer generelt i henhold til kompatibilitet med parallell utførelse på en GPU, nøyaktighet i segmenteringen og pålitelighet. En passende metode blir valgt og implementert basert på diskusjonen.

Resultatene viser at algoritmen greier og segmentere store deler av koronararteriene. Men resultatet er sårbar for bildestøy og bildefeil, og uregelmessig årefasong. Den mest signifikante forbedringen til denne algoritmen ville vært en metode som velger ut hvilke årer som skal segmenteres ut basert på deres avstand til hjerte. Algoritmen implementert i denne oppgaven presterer og produsere en senterlinje, og en segmentering på rundt 10 minutter.

Metoden som er brukt i denne oppgaven er en generell tilnærming til åresegmentering, som er spesielt justert for og segmentere koronararteriene. Metoden kan potensielt bli brukt til og oppdage og segmentere andre åre strukturer ved framtidige prosjekt.

Acknowledgments

The author wish to express gratitude for the guidance and help from the advisors of this project, Frank Lindseth and Erik Smistad. Their knowledge and expertise has been invaluable through out this project. Great thanks also goes to St.Olavs hospital and the cardiologist who work there for their input and acquisition of the CTA images used.

Contents

List of Tables

List of Figures

Abbreviations

CAD	=	Coronary Artery Disease
CVD	=	Cardiovascular Disease
CT	=	Computed Tomography
CTA	=	Computed Tomography Angiogram
ICA	=	Invasive Coronary Angiography
US	=	Ultrasound
FFR	=	Fractional Flow Reserve
CPU	=	Central Processing Unit
GPU	=	Graphics Processing Unit
ECG	=	Electrocardiogram
SIMD	=	Single Instruction, Multiple Data
SPMD	=	Single Program, Multiple Data
VRAM	=	Video RAM
MPI	=	Message Passing Interface
API	=	Application Programming Interface
GLSL	=	OpenGL Shading Language
HLSL	=	High Level Shader Language
TDF	=	Tubular Detection Filter
NLM	=	Non-Local Means
HU	=	Hounsefield Unit
GVF	=	Gradient Vector Flow
GS	=	Gaussian Smoothing

Chapter 1

Introduction

1.1 Background and Motivation

Coronary artery disease has one of the highest mortality rates in Norway, and the world. Coronary artery disease is a subclass of diseases from Cardiovascular disease. Currently it is estimated that Cardiovascular diseases account for 17 million deaths globally per year, and this number is expected to increase to 23.6 million by 2030[31]. Of these 17 million deaths, coronary artery disease was the most common cause, accounting for 8.14 million deaths globally[51]. At St. Olavs Hospital in Trondheim the standard procedure to diagnose coronary artery disease is an initial screening with a non-invasive CT angiography (CTA), followed by an invasive coronary angiography (ICA) if the initial CTA indicate that coronary artery disease might be present.

ICA is considered to be the gold standard to determine the presence, severity and location of coronary artery disease. The measurement used to determine coronary artery disease is fractional flow reserve (FFR). FFR is measured by inserting a pressure wire at the entrance of the coronary arteries to determine the maximal coronary blood flow through that specific artery. This estimate for maximum blood flow is then compared to a hypothetical case where the coronary arteries are healthy.

The Norwegian University of Science and Technology (NTNU) and St. Olavs Hospital in Trondheim, Norway has launched a joint project to aid in the diagnosis and detections of coronary artery disease. This project involves performing a FFR analysis on the initial CTA, to further determine if a patient is at risk of having coronary

artery disease. Since the initial CTA does not involve a pressure wire, a maximal blood flow measurement cannot be conducted in the normal way. Computational FFR is an alternative approach to estimate the maximal blood flow in the coronary arteries. This method only utilizes the initial CTA to produce an estimated value for the maximal blood flow. One of the components required to perform a computational FFR is an accurate segmentation of the coronary arteries. This segmentation can be done manually, but the process of manual segmentation is a very time consuming process. To make computational FFR practical in a medical environment, the segmentation needs to be fully automatic, and to be produced quickly.

Many different methods for segmenting the coronary arteries exists in literature. Common for most of these methods is that they are very computationally expensive, and require a long time to produce an segmentation. In this thesis we wish to utilize the computational power of graphical processing units (GPUs) to reduce the processing time of one of these methods. GPUs are ideal for processing large amounts of data parallel computations where many of the computations require the same instructions. We will utilize a framework *FAST*[43], which is based around the Open Computing Language (OpenCL) to implement our solution.

1.2 Project Goals

The purpose of this thesis is to explore Coronary Artery centerline extraction and segmentation, and to implement a program that performs these tasks while utilizing the computation power of graphical processing units to speed up the calculations. The main goals of this projects are:

- Explore state-of-the-art methods for Coronary Artery segmentation and centerline extraction
- Determine which methods that are applicable for this project, with respects to:
  - Potential performance increase by utilizing a GPU
  - Opportunity to make the method fully automatic
  - Accuracy of potential methods
- Implement one of these methods in *FAST*[43], using OpenCL and document the results.

1.3 Outline

The following is an outline for this thesis.

Chapter 2 - Background

In the background chapter a background study will be conducted. This will involve an introduction to the anatomy, terminology, and medical practises related to the coronary arteries. We will cover how the framework *FAST*[43] functions, theory on parallel programming, and how the imaging technology utilized to produce CTA images work. Finally, we will explore many different approaches to coronary artery segmentation and centerline extraction, and discuss and choose one of them to lay the foundation of our implementation.

Chapter 3 - Methodology

Chapter 3 goes in-depth on the method chosen in chapter 2. Each part of the method is explained and elaborated upon and implementation details, as well as pseudo code for the relevant parts is presented.

Chapter 4 - Results

The results chapter presents images of the extracted centerlines and segmentations produced by our implementation. Here we also present evaluation statistics, and run-time measurements for both available datasets.

Chapter 5 - Discussion

In chapter 5 we discuss the performance, quality, strengths and weaknesses of our implementation and the extracted centerlines and segmentations.

Chapter 6 - Conclusions

The final chapter contains conclusions for this project, and suggestions for future work.

Chapter 2

Background

In this chapter a background study will be conducted. This study will include an introduction to the anatomy of the coronary arteries, how the images used in this project are created, how coronary heart disease is normally diagnosed, parallel and GPU computing and a review of different coronary vessel tree segmentation and centerline extraction methods.

2.1 Coronary Artery Disease

Coronary artery disease(CAD) is a group of diseases under the larger group cardiovascular diseases(CVD). The most common CADs are stable angina, unstable angina and myocardial infarction and together they are the most common CVDs[4, 49]. The risk of death from CAD has decreased from 1980 to 2010 especially in developed countries[23], but despite this CAD is still on of the most common cause of death globally with around 7.4-8.1 million deaths annually[51, 34]. In general over any age, men are more prone to getting CAD, especially at ages over 65[4, 49]. The symptoms for CAD is often unclear and difficult for the patient to asses. The most common symptom for stable angina is chest pain that occurs regularly with physical activity or after other activities that strain or pressure the heart, this is associated with narrowing of the coronary arteries. The most common symptom for unstable angina is also chest pain, but when it is unstable the pain can change in intensity, frequency or character[4, 49]. It is estimated that about 30% of adults who go to the hospital with an unclear cause of chest pain, has pain due to CAD[4]. There are several risk factors associated with CAD, so in the absence of clear symptoms the risk factors are used to estimate the likelihood of CAD and to determine future investigation or treatment. The most common risk factors include family history, obesity, diabetes, smoking, lack of exercise, stress, high blood pressure, and high

blood lipids[4, 51, 23, 34]. Smoking is by far the most telling non-inherited risk factor, as about 36% of CAD patients are current or previous smokers[4]. Obesity is also a significant factor, as 20% of CAD patients are reported as obese[4].

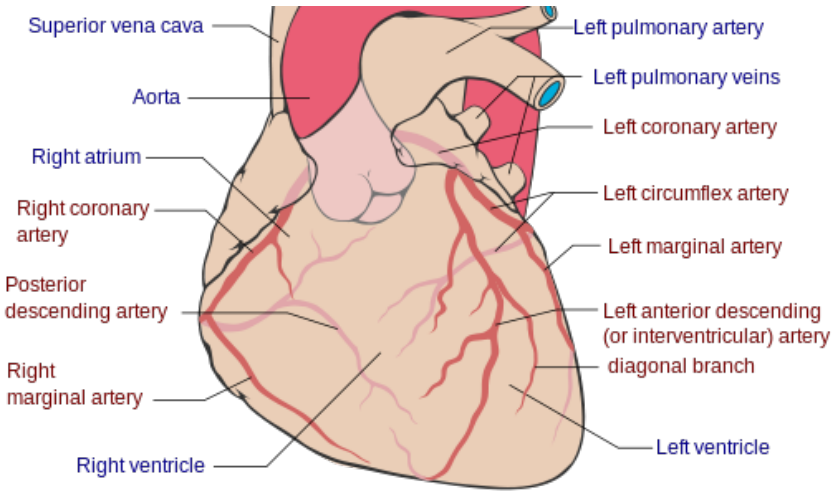

Figure 2.1: Anatomy of coronary arteries and heart[13]

2.1.1 Anatomy

The coronary arteries run on the surface of the heart and their function is to supply blood to the heart muscle. Like all other muscles the heart muscle needs oxygen-enriched blood to function and oxygen-depleted blood to be carried away. There are two main coronary arteries, the left main and the right coronary arteries[16]. The right coronary artery supplies blood to the right ventricle, the right atrium and the sinoatrial and atrioventricular nodes. The right atrium is one of two blood collection chambers in the heart, the right ventricle is one of two ventricles in the heart. The ventricles are responsible for pumping oxygen-depleted blood to the lungs[16]. The right coronary artery branches of into smaller branches, including the acute marginal artery and the right posterior descending artery. The left main coronary artery supplies blood to the left heart muscle, the left atrium and the left ventricle[16]. It then branches of into two branches, the left anterior artery and the circumflex artery. The circumflex artery encircles the heart muscle and supplies blood to the outer and back side of the heart. The left anterior artery supplies blood to the front and left side of the heart[16].

2.1.2 Diagnosis

There are several ways to diagnose CAD. Some of these methods are not definitive on their own, but they can provide an indication or reinforce the suspicion that a

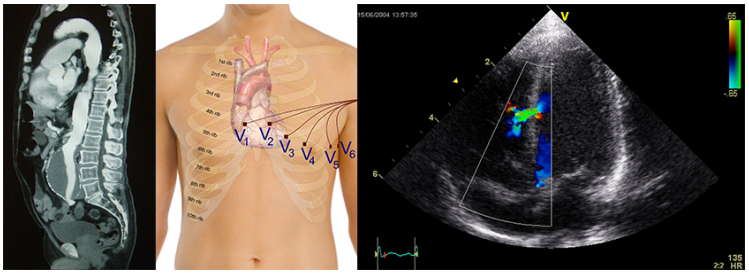

Figure 2.2: Abdominal aortic aneurysm[12], Electrocardiogram electrodes placement [14], Echocardiogram Ventricular spatial defect[11]

CAD might be present. The most common methods are CT angiogram, echocardiogram, electrocardiogram and stress test. Following is a short explanation of what each of them are:

Electrocardiogram: An electrocardiogram (ECG) records electrical activity in the heart using electrodes placed on the patients body. The electrodes record tiny electrical variations on the surface of the skin that arise from the heart muscle during each heartbeat.

Echocardiogram: Is using ultrasound on the heart. If any parts of the heart is moving weakly/irregularly it may indicate that these parts is receiving too little oxygen.

Stress test: A stress test is in this instance a stress test of the heart. This can be done in several different ways, and can involve several kinds of additional measurements. It can simply be to see if the patient experiences pain during heavy heart activity. Or it can involve etc echo/electrocardiogram, to get more information from the test. This test alone in not enough to diagnose CAD, but it might reveal a problem and give an indication for future testing.

CT angiogram(CTA): Is a heart imaging technique to determine if plaque buildup is obstructing the blood flow in the coronary arteries. To see the blood flow on a CT image contrast material is required. This contrast material is injected in two portions, the first to establish how long time the contrast material injection needs to travel to the coronary arteries, the second injection is then for the actual angiogram.

Cardiac catheterization: This is not a test on its own but rather a technique where a flexible, long, thin tube called a catheter, is inserted into a blood vessel in the arm, upper thigh or neck and threaded to the heart. When the catheter is in place, a vari-

ety of tests can be conducted. It can be a delivery mechanism for contrast material during a CT Angiogram, or it can be used while an echocardiogram is conducted to detect blockages or obstructions in the coronary arteries. It can also be used to measure loss of blood pressure because of obstructions in the arteries. This kind of testing is more invasive than the other alternatives but it has a much higher chance to successfully detect and diagnose CAD.

More information on these diagnosis methods can be found here:[4, 49, 16]

2.2 Imaging

This thesis will be about segmenting the coronary arteries from CT images. The CT imaging technology, what it shows, and how those images are generated play a major part in how the algorithm for the coronary artery segmentation is designed. This chapter will cover a simple, general explanation of how CT imaging works from a practical standpoint.

2.2.1 Conventional X-ray

A CT scan uses a combination of many X-ray images taken from different angles to create a cross-sectional image. Therefore to properly explain how CT works, we need to explain X-rays. X-rays are created by shooting a plate of metal with electrons. The electrons will interact with the atoms in the metal and photons will be emitted with a very high frequency. The photons are then directed at a body. The denser material in the body will absorb most of the photons, while the less dense material will absorb less. As the photons pass through the body they hit a film behind the body. Locations where the photons hit will become darker, and as more photons hit the same locations the darker that location becomes on the film. The film becomes the X-ray image.[1]

Some materials in the body absorb more of the photons than other. Bone and teeth absorb a lot, meaning that the areas on the final image that are almost completely white are bone or teeth. The other, less dense parts form shadows or silhouettes on the image. Because of this, X-rays are usually used to examine teeth and bone as the image simply doesn't provide enough contrast on the rest of the tissue to provide useful information.[1]

2.2.2 Computed Tomography

An X-ray computed tomography (CT scan) uses computer-processed combinations of several X-ray images taken from different angles to create a 3D image. By doing this some of the disadvantages with conventional 2D X-ray images are diminished. Meaning that CT can display more than just shadows and it is better at displaying contrast between different soft-tissue. Because of the nature of multidirectional X-rays, CT can image the absorption of each location in a 2D slice of the body. This is an advantage over X-rays as it can only display the sum of all absorption at a specific direction through the body. By interpolating all the different X-ray signals in the frequency domain, a 2D frequency image can be collected. The actual image this produces can be retrieved by applying the inverse fourier transform.[10]

As stated earlier, a CT scanner use several 2D slices to form a 3D volume. Each voxel at any specific location has a number indicating the amount of X-ray absorption at that location. This number is usually stored in Hounsfield Units (HU). The Hounsfield Unit scale is a linear transformation of absorption value and is defined so that distilled water at 1 bar and 25C has 0 HU. Below is the equation that defines the HU scale and a table that show a list of common substances in the body.[47]

$$HU = 1000\frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{2.1}$$

Substance	Air	Lung	Soft Tissue	Water	Blood	Muscle	Bone
HU	-1000	-700	-200	0	30-45	40	700-3000

Table 2.1: Table of HU values for some substances

2.3 Parallel and GPU Computing

Parallel computing is the practise of performing several calculations simultaneously. This builds on the principle that large or complicated problems often can be split into smaller sub-problems[30]. These sub-problems are then solved in a parallel manner to solve the overarching problem in less time than with conventional serial computation. Parallel computation has been relevant in high performance computation for several years, but for lower end, consumer grade systems it was largely

unused until the introduction of multi-core CPUs in the early 2000s[30]. Since then more and more applications and programs have started to utilize parallel computing to increase performance. Several different forms of parallel computing/parallelism exist, but they are usually divided into two main groups: Task parallelism and Data parallelism.

2.3.1 Task Parallelism

Task parallelism, also known as function parallelism is based on the idea of dividing a program into separate tasks and then run them simultaneously on different processing elements. These tasks, or subprograms can be very different from each other. Meaning that there are no specific code or complexity requirements to enable task parallelism. The different subprograms usually need to communicate with each other, and to perform synchronization between each other to ensure data validity.

There are two main categories of sub-problem communication, shared memory and messages. Shared memory is when two or more of the tasks share a certain portion of the memory available to them both or all of them. To make this work without synchronization issues, some sort of access or validity system to make sure that none of the tasks are working with false data needs to be implemented.

Messaging between tasks is often used for larger distributed memory machines. Since their memory is distributed on a per machine or per component basis, having shared memory is unpractical. Therefore a message passing protocol is more practical. An obvious downside to performing synchronization and management through messages is the speed, but in the case where shared memory is not a viable option, and/or the different tasks are large enough to warrant a large system (supercomputer) it is often the best alternative.

Shared memory is done by giving all, or the relevant tasks access to a shared portion of the available memory. While this practice is fairly simple, making sure that the data in the shared memory is valid is a major problem. Since several tasks can both read and write/overwrite invalid or false data can be a problem. There are several solutions to this problem, the most common solution concepts are barriers and flags. Barriers block tasks until all tasks have reached a predetermined point. Flags enforce that only one task can modify the shared memory at a time.

2.3.2 Data Parallelism

Data parallelism focuses on distributing data to different computing nodes. In general the same task is performed on the distributed data in parallel on the different computing nodes. This form of parallelism works well with tasks where the data is independent, meaning that one part of the data does not depend on any other part of the data. With for example image processing, where each pixel, or voxel is processed independently from any other pixel or voxel data parallelism works well.

SIMD (Single Instruction, Multiple Data) is a term from Flynn's taxonomy which is recognized as the simplest form of data parallelism. With SIMD a single instruction is performed on each element in the data. This is largely used as a concept rather than actual practice when it comes to parallel programming. A more practical approach is to incorporate classical procedural programming with SIMD. This is referred to as stream processing, and it take the concept of SIMD but instead of just performing one instruction it performs several, in the same way procedural programming handles instructions. This gives to possibility for higher efficiency as it can allow higher arithmetic intensity compared to vector processing where data is read and stored per instruction. As some instruction require more time than others, stream processing can enable a more strategic approach to parallel programming. The set of instructions that are applied to each part of the data is usually referred to as a kernel.

An additional variation of stream programming is SPMD (Single Program, Multiple Data), this incorporates the stream processing approach, but also includes the option to use branching. Branching is typically done by if-else statements, and it opens up the possibility that some of the data skip or include instructions, meaning that all the data parts does not necessarily perform all the instructions.

2.3.3 Parallel architectures

Parallel computation requires hardware architectures that enables parallel computation execution. There are 4 general layers of parallelism in modern hardware, multi-chip, multi-core, multi-thread and instruction. Modern processors use a combination of these types of architectural parallelism.

Multi-chip: Several physical processor chips

Multi-core: Multiple processing cores on the same chip

Multi-thread: Two or more threads on one processing core, that can be switched in between with very little overhead and delay. This enables the processor to reduce idle time.

Instruction: A processing core that can perform more than one instruction per cycle.

CPU - Central Processing Unit

The CPU is the traditional processor found in computers, tablets, phones and so on. It is the main processing unit in traditional systems. As stated earlier, traditionally programs were executed serially on a single CPU, meaning that each instruction was executed in a strict order. An internal clock, usually referred to as cpu cycle, is used to control the rate in which instructions are computed. This clock, or cycle, is also integral in synchronizing the various components in the computer. In general, CPUs were made faster by increasing the amount of transistors, and by increasing the clock frequency. To increase the frequency and the amount of transistors the voltage has to be increased. This leads to an increased power consumption, and while that might be manageable, the heat that increased power consumption produces is a problem. Around 2004, Intel's CPUs throughput reached roughly 4 GHz. This proved to become an estimated limit of what is practical to handle with conventional cooling systems.

Because of this limit, the focus in CPU manufacturing changed from increasing processing speed on a single core into increasing processing speed by using parallelism. As mentioned earlier, there are several approaches to parallelism that is viable for different purposes. While supercomputers can do well by using several connected computers in a multi-chip system, this is not viable for consumer grade computers. A multi-thread and Instruction parallelism approach was and is effective, but in itself it was not enough to provide a solid base for future development in processing speed. The focus turned to multi-core processors with multi-threading and Instruction parallelism. This made the CPU capable of processing several different programs at the same time on the same chip. At the time of writing, CPUs in the commercial market usually have four, six, eight or even sixteen cores.

GPU - Graphic Processing Unit

A graphic processing unit, also known as a visual processing unit, is a specialized processor. GPUs have several purposes, but as the name suggests, the primary focus is to process graphical content. GPUs does this very well compared to the more general purpose CPU, as the calculations the GPU is intended for areas like texture mapping, polygon rendering and coordinate transformations. All of these operations can often be very memory intensive, promoting the trend of including more and more integrated memory to the GPU, usually referred to as VRAM or Video Ram. The nature of these operations in a graphical setting is also very uniform, meaning that in many cases the same set of instructions is to be performed on each individual member of data. That makes the typical graphical operations highly data parallelizable, and thus the development on GPUs over the years have been to promote the parallel processing speed, by including more functional units, more memory and so on. Therefore GPUs are typically a type of SIMD processor. This means that a GPU is optimal for performing the same sets of instructions on a collection of data in parallel. A GPU is better at this than an CPU because it typically has hundreds of functional units. A functional unit is sometimes referred to as a "core", in the same way a CPU might have multiple cores. But this is not entirely accurate, as McCool et al[30] defined a core as a processing element with independent flow control. This is not true for the individual functional units on the GPU. Instead the functional units on a GPU is grouped together, meaning that the group of functional units all has to perform the exact same instruction in the same cycle. Each of these groups can be referred to as a core, as they do fulfill McCool et al[30] definition, but that might be easily misunderstood when considering that "core" is usually used for the individual cores on a CPU.

GPUs usually comes in two forms, dedicated, or embedded/ integrated. The embedded/integrated type of GPU is usually on motherboards. These are typically a lot slower than dedicated graphics card. The dedicated GPU is a standalone card, this allows for much more space for both additional functional units, more memory, and dedicated cooling.

Most GPUs use the SIMD variation SPMD, meaning that they allow branching. While this is practical and might make programming for the GPU easier, it does create the need for both knowledge of how to use branching effectively, and the necessary functionality in GPUs to handle when the branching is inconsistent, or

divergent as it is typically named. If the instruction flow, or code flow for all the execution threads in one SIMD group follows the exact same path with the same branching, the code flow is convergent. When this is the case, the SIMD group can simply process the data with no special treatment. But if this is not the case, if the code flow is divergent, the GPU has to run all the different paths that are present within the SIMD group, for all members of the group. That means that even if one particular code flow on one particular functional unit does not branch or avoids some branches entirely, it will still have to do it. The result of this is that the time saved by performing parallel computation is lowered, or even negative. The GPU will still be able to produce the correct result though, as it has masking techniques to ensure that the unnecessary code that is run does not affect the final result.

2.4 FAST

The purpose of this thesis is to implement an segmentation algorithm for segmentation and centerline extraction of the coronary arteries. This is to be implemented within an already existing framework named *FAST*. *FAST* stands for Framework for Heterogeneous Medical Image Computing and Visualization[43] and is developed by Erik Smistad, Mohammadmehdi Bozorgi and Frank Lindseth. FAST primarily focus is on computing medical images by utilizing the computational power in GPUs. There are several benefits to implementing the algorithms required for this thesis in FAST, it has functionality for visualization, a standardized execution pipeline which helps with memory management and synchronization, and many of the complementary tools needed to perform this kind of segmentation is already present.

As the focus of FAST is to execute the necessary image processing algorithms on a GPU, we will elaborate on some of the tools needed to exploit the parallel architecture for general purpose programming on GPUs. This will also involve some of the basics theory behind programming for parallel execution.

2.4.1 Parallel programming

In order for programs to exploit parallel execution, the CPU/GPU has to know what it can run in a parallel manner. There has been and there are still compiler and processor designers who tries to do this automatically. But so far, adapting serial code to perform in parallel automatically, has largely been unsuccessful. To effectively

run and write code for parallel architectures, explicit parallel languages, libraries, and APIs are needed.

Message Passing Interface (*MPI*) is such an API. MPI is primarily used for writing parallel code on machines with distributed memory, such as supercomputers. POSIX Threads (*PThreads*) is a more basic API for creating and managing threads that run on the CPU. PThreads in itself only gives the programmer the capability to use threads, it does not aid in synchronization or data management, therefore this needs to be done by the programmer. OpenMP is another popular parallel programming API. It covers more than PThreads in many ways, but it is more restrictive in that it only works on shared memory systems. There are many other APIs out there, like Linda, TCGMSG an alternative to MPI, PVM another one but designed for networks, and many more.

There is also programming languages especially designed for parallel programming. These were made as many of the older, and more popular programming languages were not made with the express purpose of making parallel programming easy, or understandable. Therefore, some programmers believe that new programming languages would be better suited for the task of parallel programming. X10 is one example of such a language, it is developed by IBM and is heavily influenced by Java and C++ in the effort to make it easy to learn for developers. Chapel is another one, and is designed with the goal of increasing supercomputer productivity, but it also works on smaller consumer grade systems.

GPUs and CPUs can be capable of doing the same thing, but programming for them is sadly not the same, at least until the introduction of languages like OpenCL. Solving a general purpose problem on a GPU was originally done by using shader programming. Shader programming refers to using certain parts of the rendering pipeline found in GPUs. There are several programming languages for this, like GLSL, also know as OpenGL Shading Language, HLSL (High Level Shader Language) and C for Graphics (Cg). But none of these languages are capable of solving the general problem with shader programming. To solve general purpose problems with shader programming the problem has to be transformed into a graphics rendering problem, and this has in many cases proven to be a difficult and frustrating task. Therefore new solutions, not only from external users but also from the GPU producers were introduced. CUDA is a language/API developed by NVIDIA for

general purpose programming on NVIDIA GPUs. This language made it possible for a developer to exploit the power of NVIDIA's GPUs without first transforming a problem into a rendering problem. A few years later OpenCL was released, this was inspired by CUDA and does many of the same things. It is much more inclusive, in that it can be used for parallel programming on GPUs from several producers, on CPUs and other parallel architectures.

2.4.2 OpenCL

Open Computing Language[22, 24] is a framework for writing programs that can run on both CPUs and GPUs. More specifically, OpenCL is made to make programming for heterogeneous platforms easier. OpenCL is ratified and maintained by the Khronos group[24] but it is up to the GPU producers to make drivers and compilers for OpenCL so that OpenCL code can run as specified in the standard on that specifics producers hardware.

OpenCL consists of two parts:
OpenCL C Language - Is used to write kernels, and is effectively an extended version of C.

OpenCL Runtime API - An API that is used to synchronize and control the different devices available on the machine.

OpenCL is modeled to have a single host that distributes work to several devices. The main distributer/executer is a thread running on the CPU. The additional devices can be accessed through the OpenCL API, where the host can perform queries towards these devices to get information about them. These queries are important, as they make a difference in how these devices are programmed towards. Since the available devices differ from system to system these queries are important, as information like the number of compute units, or max clock frequency can make a difference when programming for those devices. It is also the host's responsibility to compile kernels for each specific device. These kernels are written in the OpenCL C language and they are command/instruction queues. As mentioned earlier, OpenCL supports both task and data parallel execution, where the data parallel model used is SPMD.

When programming for a GPU OpenCL uses a NDRange hierarchy. An NDRange can be in up to 3 dimensions and all the potential dimensions can be set. The NDRange is then divided into work-groups based on the dimensions set. These work-groups are divided again into work-items, and it is these work-items who execute the kernels. Each kernel has a global-id, group-id and a local-id which is an N-vector. All the work-items in a work-group run on the same compute unit. For

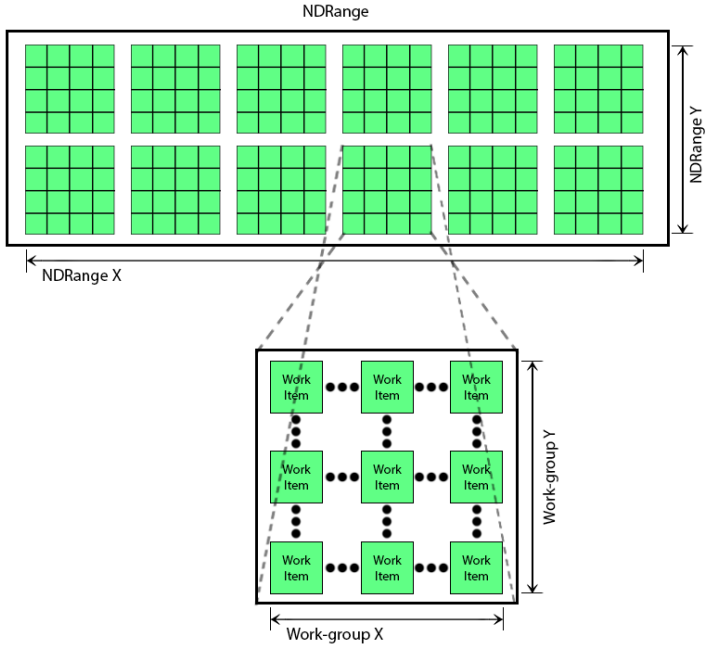

Figure 2.3: OpenCL execution model - NDRange

kernels written for GPUs in OpenCL C can access four different memory levels. Each of these has different parameters for how they are shared, physical location, speed and size. The largest, but slowest memory is the global memory. This can be accessed by any work-unit and can be both written to and read from. The second level is the constant memory. This memory works the same way as global memory does, but it is smaller and read-only and that means that it can be accessed faster. The third level is the local memory. This is accessible for each member of a work-group. It is faster than the constant memory, but is usually quite small in size. The fourth and final level is the private memory. The private memory is private to each work-unit and is the fastest memory available, but is also the smallest in size.

2.4.3 Execution Pipeline

In addition to OpenCL, the execution pipeline FAST utilizes is important to how both algorithms and additional components are implemented. The purpose of the

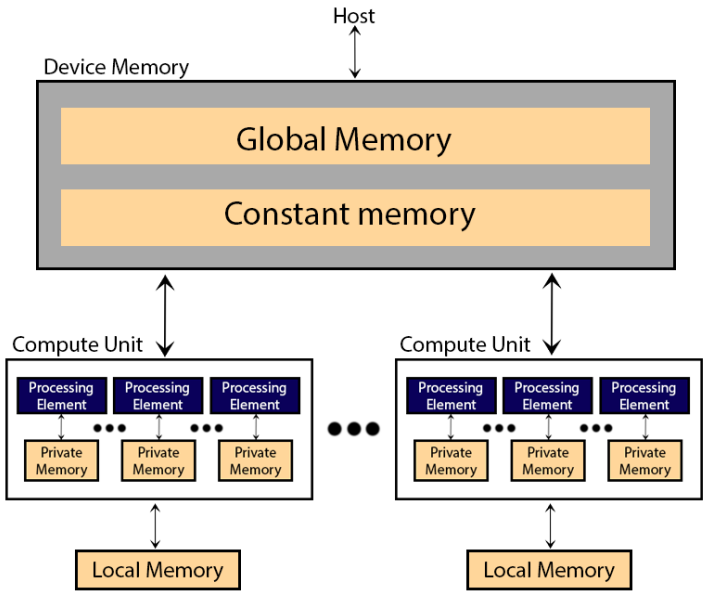

Figure 2.4: OpenCL memory model

Memory	Access	Read/Write
Global	All	Read/Write
Constant	All	Read only
Local	Work-group	Read/Write
Private	Work-unit	Read/Write

Table 2.2: Table of GPU memory access

execution pipeline is to make sure that each individual component in a larger sequence is executed in the correct order. A program will probably have several components, like initial gui creation, pre computation, execution on both the CPU and GPU, and more. It is important that these components are executed in the correct order to avoid synchronization issues like race condition or missing/unavailable data. The execution pipeline in FAST consists of ProcessObjects. These are parent objects that are linked together to form the pipeline.

2.4.4 Data and access management

A ProcessObject can have DataObjects connected to it. DataObjects, as the name implies controls the data any FAST implementation uses. The DataObjects, and the connection between a particular DataObject and ProcessObject ensures that synchronization errors like race-conditions and data validity doesn't occur. Each DataObject has an internal time-stamp, this time-stamp is updated each time data in that particular object is changed. The connections between a DataObject and a Pro-

cessObject also has an time stamp, this indicates which version of the DataObject was used the last time the ProcessObject was executed. A DataObject can contain

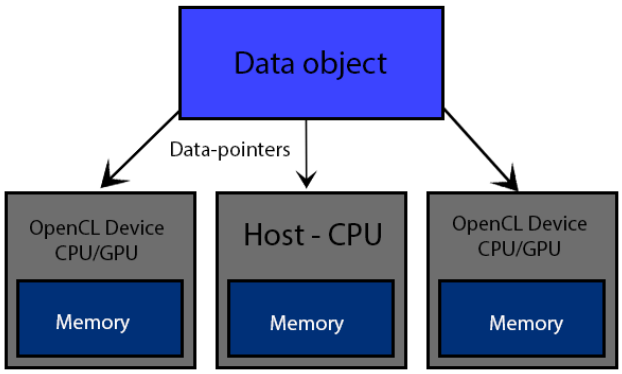

Figure 2.5: FAST memory management

pointers to data on several devices, and in some cases the same sets of data can be on different devices. Image and Mesh data are two of these cases, where the same data can be in several different devices memory. This can be an error source, as in when data is changed one device it no longer matches with the data on other devices. FAST handles this by changing the affected data according to the original change on the device where the change occurred. This ensures that the data, usually an Image or a Mesh is the same on all devices.

2.4.5 Visualization

There are several renderers implemented in FAST. These are implemented with the renderer object, which has ProcessObject as a parent class. Therefore the renderer follows the execution pipeline, and the rules that it implements. FAST has also integrated Qt. Qt is a well known framework and library used to develop application software. In FASTs case, Qt is primarily used to develop GUIs. FASTs renderers work on both 2D and 3D images, and is specialized to handle medical images. This is a large benefit of using FAST, as implementing custom renderers is both time consuming and difficult.

2.5 Coronary Artery Segmentation

In this thesis the main goal is to produce an accurate segmentation of the coronary arteries with only a CTA image as input. A segmentation is a labeling of each voxel in the CT image that determines if the voxel is a part of the coronary arteries or not. There are two notable reviews on this topic by Kirbas et al[25] and Dehkordi

et al[15] respectively. They both cover many of the same issues and reach a similar conclusion.

In the review by Dehkordi et al[15] they divide the different approaches of coronary artery segmentation into pattern recognition, model-based tracking and propagation, neural network, fuzzy, and artificial intelligence based methods. They cover a large number of suggested approaches within each method field, and discuss strengths and weaknesses based on the presented results.

While the review by Dehkordi et al[15] focuses solely on coronary artery segmentation, a review by Kirbas et al[25] focuses on vessel segmentation in general, and applies them where applicable to the coronary arteries. The review by Kirbas et al[25] is larger and more in-depth then the review by Dehkordi et al[15]. They cover a large array of methods, and in a similar fashion as Dehkordi et al[15], they divide the different methods into categories. The categories presented by Kirbas et al[25] are: pattern recognition, model-based , tracking-based, artificial intelligence-based, neural network-based, and miscellaneous tube-like object detection approaches.

At the 11th International Conference on Medical Image Computing and Computer Assisted Intervention 2008 workshop *"3D Segmentation in the Clinic: A Grand Challenge II"*[7] a framework for validation and testing of coronary artery centerline extraction was presented. This framework consisted of testing software, and 30 coronary artery centerline extractions done manually by professionals and validated by professionals within the field. This eventually got the name *Rotterdam Coronary Artery Algorithm Evaluation Framework*[17]. This framework allows for scoring and validation of coronary artery centerline extraction algorithms on independent data, and the algorithms tested are all ranked against each other with public scores. This framework has stayed relevant since the release in 2008, and is still being used today. Many of the algorithms that are currently ranked high where released after 2008.

In this chapter we will cover a basic introduction to the most common approaches to coronary artery segmentation and centerline extraction. The review by Kirbas et al[25] is used as the primary source of categorization, with the review by Dehkordi et al[15] supporting, particularly within the artificial intelligence, machine learning

and neural network categories. The ranking from MICCAI'08[7, 18] will provide the final up to date reference for centerline extraction algorithms.

2.5.1 Pattern Recognition

The review by Kirbas et al[25] and the review by Dehkordi et al[15] covers a large array of pattern recognition based methods. Kirbas et al[25] further divides Pattern Recognition into several categories: Multi-scale, Skeleton-based, Region growing-based, Ridge-based, Differential geometry-based, and Matching filters-based approaches. Some of the methods covered by Kirbas et al[25] are covered by Dehkordiet al[15] to. But since the review by Kirbas et al[25] was conducted in 2003, and the review by Dehkordi et al[15] was conducted in 2011 most of the approaches covered by Dehkordi et al[15] are slightly more refined, but still follows the same concepts.

Multi-scale-based Approach:
Multi-scale approaches performs the segmentation at several image resolutions. The main advantage and motivation behind this technique is increased processing speed. Large structures are extracted from a low resolution representation of the original image and fine or smaller structures are extracted from either the original image or an enhanced version. This general approach also offers another advantage in that since the extraction of structures normally is done at least twice, there is a greater chance that one or more of the iterations discover the same structures. And when the same structure is detected more than one time they can both/all be used to refine the final result.

The multi-scale approach is usually teamed with some other method of vessel detection and/or segmentation to perform the actual segmentation, as the multi-scale technique is used primarily to decrease processing time. Kirbas et al[25] referrers to two implementations where the multi-scale approach is used: One is a 3D coronary artery centerline extraction based on three resolutions where they use linear programming and relaxation-based labeling on each resolution and then match and extract from all three resolutions. The second one uses a multi-resolution analysis based on wavelet transformation. Their implementation also aims to detect flow direction and flow volume and they use velocity-sensitive, phase contrast MR images.

Skeleton-based Approach:
Skeleton-based methods extract centerlines and then use them to perform 3D reconstruction. Both Kirbas et al[25] and Dehkordi et al[15] covers some methods where centerline extraction is used.

A method by Niki et al[33] uses a short scan cone-beam filtered back-propagation reconstruction algorithm aided by a graph description of the blood vessels to extract the centerline and then uses thresholding and an object connectivity procedure to perform the segmentation.

Another method where the goal is to segment airways in the lungs uses thresholding, then a 3D thinning algorithm to extract the centerlines.

A more complex approach by Sorantin et al[44] is presented where a five step algorithm is applied to a CT angiogram. Laryngo-tracheal tract(LTT) is extracted using fuzzy connectedness based on a user-supplied seed points. 3D dilation is applied to avoid uncertain boundary points due to partial volume effect. The resulting 3D volume is then converted into cubic voxels based on interpolation. The second step is then to apply a 3D thinning procedure on the volume from the first step. The third step is to use a shortest patch searching algorithm on the thinned volume from step two. This step requires the user to manually input start and endpoint on the estimated central path. The forth step is to smooth the result, and the final step is to calculate a cross-sectional profile along the medial axis of the smoothed result.

Both Kirbas et al[25] and Dehkordi et al[15] reviews several more methods where a skeleton-based approach is used. Many of them use methods that are fairly similar to the first and second method presented here.

Region growing-based Approach:
The region growing-based approach starts from one or more seed points, and iteratively/incrementally add neighbouring pixels or voxels to a the segmented region based on some predefined criteria. It is based on the assumption that pixels or voxels that are close to each other and have similar intensity values belong to the same object. The main disadvantage to region growing-based methods is that they often

requires user-input for the seed points. Because of how the region grows incrementally it is also vulnerable to noise and image artifacts as these can lead to holes or leaking (over-segmentation). Both Kirbas et al[25] and Dehkordi et al[15] reviews methods based on the region growing -approach.

O'Brien and Ezquerra[35] uses a region growing approach to extract centerlines of coronary arteries from CT images. Their algorithm start with a low pass filtering of the initial image as pre-processing. Then an initial segmentation by region growing is performed based on user provided seed points. After that the centerlines are extracted from this initial segmentation by employing a balloon test. Undetected vessel segments will then be located by a spatial expansion algorithm. Then graph theory is applied to determine which centerlines to include in the extraction and which centerlines to ignore.

Both Kirbas et al[25] and Dehkordi et al[15] reviews several more methods based on region growing. Common for all of the methods covered is that they require user input for the seed points, and that they use these points to either generate a centerline, or to directly segment the target vessel based on the supplied seed points.

Ridge-based Approach:
Ridge-based approaches uses the "ridges" created by intensity or gray-scale or some other image characteristic mapping. The general technique uses intensity from a gray-scale image as a 3D elevation map. The method builds on the assumption that the local maxima in this elevation map are points in the segmentation/centerline. These local maxima points can be used as a representation of the centerline alone, or they can be used as seed points where the ridges are traversed based on the seed points. Usually, methods which use the ridge-based approach generates a centerline, therefore the methods presented here can also be classified as skeleton-based approaches. Both Kirbas et al[25] and Dehkordi et al[15] reviews several methods who utilize a ridge-based approach.

One of the main methods where the ridge-based approach is utilized is a method by Bullitt and Aylward[3]. Their method relies on manually selected seed points for each vessel to be extracted. An intensity ridge map is constructed and for each seed the closest ridge is traversed. This results in a line of points generated from

the traversal and for each point and estimate of the diameter of the vessel can also be calculated based on the intensity map.

Aylward et al[2] uses the same technique described earlier but with further refinement to more accurately extract centerlines. They apply the cores method[37] on the intensity ridges. From manual seed points they locate the ridges to be traversed by using a conjugate directions search with respect to the hessian matrix.

Differential geometry-based Approach:
Differential geometry based approaches treat images as hypersurfaces, and extract features from these using curvature and the crest lines of the surface. This technique is based on the assumption that the crest points of the hypersurface corresponds to the centerlines of the vessel structure to be segmented. This method works for both 2D and 3D images, and they are modelled as 3D and 4D hyperstructures respectively. For a 3D image the generated surface can be described by two principal curvatures and by their corresponding orthogonal directions. The orthogonal curvatures correspond to the eigenvalues of the Weingarten matrix and the orthogonal directions are the eigenvectors. The local maxima of the maximum curvature given by the hyperstructure are used as link points to create a centerline structure. Kirbas et al[25] reviews several methods where differential geometry is used.

A method by Krissian et al[26] is reviewed where they use a *Directional Anisotropic Diffusion* method derived from Gaussian convolution to reduce the image noise. Their method is based on the differentiation of the diffusion in the direction of the gradient, minimum, and maximum curvatures. One of the primary benefits with the *Directional Anisotropic Diffusion* method used is that it effectively reduce noise, without introducing significant blurring. This method is applied to a set of phantom images, and produces an image where the vessels are significantly enhanced and suited for some other extraction algorithm.

A method by Prinet et al[39] is reviewed where they utilize the method described above. A cylindrical mathematical model is used to identify and represent the vessels from the 3D/4D image generated from the general method presented. This method requires no additional knowledge, and is fully automatic. This method generates a centerline, therefore it can also be classified as a skeleton-based method.

Matching filters-based Approach:
The matching filters approach uses as the name implies several matched filters to extract the objects in question. For vessel extraction these filters is normally differentiated by the size and orientation of the vessels to be extracted. Then after the filters has been applied some other image processing or extraction method, like thresholding to get the final segmentation and/or centerline extraction. Kirbas et al[25] reviews a few methods where the matching filters principle is applied.

A method presented by Poli and Valli[38] is reviewed where they use a set of multiple oriented linear filters obtained as linear combination of properly shifted *Gaussian* kernels to detect vessels in *real time*. The filters applied are sensitive to both different orientation and thickness of the vessels to be extracted. In addition to this they use convolution masks to obtain maximum efficiency of the vessels. And they use the orientation and scale information obtained from the linear filters to only extract vessels and no other structures. This results in a centerline, and they use thresholding with hysteresis[8] to perform the segmentation based on that.

2.5.2 Model-based Tracking and Propagation

Model-based tracking and propagation approaches attempt to apply vessel models to extract or identify vessel structure in medical images. Kirbas et al[25] divides the different model-based and propagation approaches into four general categories: deformable models, parametric models, template matching and generalized cylinders. Generalized cylinders is a spacial case of the parametric approach, and will be included in the short explanation about the parametric models.

Deformable models:
Deformable models is a generalization of methods where one or more deformable objects interact with image characteristics through internal and external forces. The active contour model or *snakes* is probably the most popular method within the deformable models category. The *snakes* method is a general method where a line or *"snake"* of connected points are affected by external and internal forces. Each of the points on the line is affected by theses forces, and by each other. The internal forces impose smoothness and connectivity to the snake. They enforce that the line does not rip apart, or bend in *curves*. The external force pull the points in the line towards the desired image features.

These two forces makes the deformable line robust to both image noise and artifacts. The main disadvantage with this model is initialization. Where the snake or snakes are positioned at initialization, or how long each snake is, make a significant difference for the end result. Each snake needs to be handled individually, but it is also necessary to avoid collision and overlap between the snakes. Therefore, it is advantageous to have a limited number of snakes active, at good positions, so that the results are satisfactory and the computational load is manageable. Kirbas et al[25] reviews several implementations of this and very similar methods.

Parametric models:
The parametric approach tries to identify objects of interest parametrically. In general, for vessels and tubular objects, and assumption is made that their general shape is a set of overlapping ellipsoids. This is normally referred to as the circular vessel model. On of the problems with this model is irregular vessel shapes. Vessel junctions, irregular vessels in general, or vessel stenoses might not fit within the ellipsoids shape assumption and negatively affect the results produced. Therefore, in practice this method is usually teamed with some other method to detect the shapes that fall outside of the overall assumption. Kirbas et al[25] reviews a few methods where this approach is used, most of these are teamed with either an artificial intelligence method or a pattern recognition method.

Template matching:
The template matching approach attempts to recognize a structure, or structures in an image. This method normally uses *a priori* knowledge normally referred to as a context or a template to match with the potential structures in the input image. For coronary arteries this context is usually a series of points or nodes that is deformed within some restrictions to fit the structures detected in the image. Kirbas et al[25] reviews a few methods where this approach is used. These methods vary in that they use different methods for the deformation to fit the objects to be identified. Petrocelli et al[36] uses dynamic programming to handle the deformation. Summers and Bhalerao[45] uses an estimation system where they estimate features like flow direction, vessel angle, and diameter to attempt to perform the deformation.

2.5.3 Artificial Intelligence

The Artificial Intelligence category is very broad. Kirbas et al[25] defines it as methods who utilize knowledge to guide the segmentation or detection process and

the delineate vessel structures. This knowledge can be general knowledge about the imaging technique used, the area captured in the image, knowledge about the patient or the patient condition, and much more. In practice many, if not all the methods covered here can be classified as AI methods. Smets et al[42] presents 11 general rules about the appearance of blood vessels. The main disadvantage Kirbas et al[25] argues is the computational complexity of most AI detection and segmentation methods. Therefore, artificial intelligence approaches are usually combined with other less complex methods to remove some of the unwanted information and reduce the computational load the AI method needs to process. As mentioned earlier the review by Kirbas et al[25] was done in 2003. Since then there has been significant development within the AI field when it comes to medical image analysis.

Several newer methods tested with the MICCAI'08[7, 17, 18] framework utilizes AI techniques and methods. Some algorithms, like the one presented by Schaap et al[41] and the one presented by Kitamura et al[55] are based on machine learning. While others, like Zheng et al[54], Szymczak et al[46], Lesage et al[29], Yang et al[53], Cetin et al[9], Krissian et al[28], and Bauer et al[6, 5] are based on expert knowledge and intelligent searching.

A method presented by Friman et al[21] utilizes a multiple hypothesis tracking approach which is complemented by a minimal path search when needed. They describe their algorithm as an *"interactive approach to the identification of coronary arteries"*. By that they refer to the fact that their method is not fully automatic. How much user interaction that is required varies depending on how well the multiple hypothesis tracking algorithm performs. But some interactions is required, both starting points and end points need to be supplied by the user. If the hypothesis tracking algorithm performs well, in that it produces a complete connected centerline network, then no more user input is required. The minimal path search then works as a backup solution, where the user can supply more start and endpoints, and the minimal path search will attempt to connect them. Finally, if the centerline network is still incomplete, their algorithm and design allows the user to manually trace the centerline.

A method presented by Zheng et al[54] uses a model-driven approach. Their algorithm attempts to first locate the general area of the coronary arteries by detecting

the heart chambers, and then attempting to place a *mean* centerline in close relation to the heart chambers. Then the *mean* centerline is transformed iteratively based on image information to fit the coronary arteries. This transformation is done by analyzing each *"point/coordinate"* in the *mean* centerline, searching for the best candidate among a field of neighbours based on a cost function, its distance from the already located best points in previous iterations, and a balancing parameter. This method utilizes machine learning in both the creation of the *mean* centerline, and in balancing the cost function, and in the weighting of distance versus cost score within the overall cost function.

Szymczak et al[46] algorithm for centerline extraction consists of two major steps. The first step is to identify *core* points. These points are centers of intensity in two-dimensional slices through the input image. Then, a weighted core graph is constructed by connecting nearby *core* points with *edges*. These *edges* are constructed by a shortest path weighted search. Then when all the *core* points have been processed, the output is the shortest path connecting the starting point and the endpoints. The starting point and the endpoints can be detected automatically (i.e *core* point with only one connection), or they can be user supplied.

Yang et al[53], Krissian et al[28] and Bauer et al[6, 5] all have strong similarities in their methods for extraction of the coronary artery centerlines. In all three methods they use variations of Frangi et al[19] tubular detection filter to detect potential points within the arteries. Bauer et al[6, 5] uses the conventional Frangi et al[19] vesselness filter, while Krissian et al[28] uses a an vesselness estimation filter similar to a circle fitting filter. Yang et al[53] vesselness filter utilizes the base of Frangi et al[19] filter, but removes points at the boundaries of the cardiac chambers.

Then the points/scores created by the vesselness filters are processed to create the centerlines. Yang et al[53] chooses the most promising points, and uses these as start points in a ridge traversal search manner. The search is supported by biomechanical metrics, to avoid errors as big and sudden direction changes, or invalid connections.

Krissian et al[28] uses a region growing approach based on the most prominent vesselness points. Where the most prominent points are calculated based on their

vesselness score, and its location relative to each other and the aorta. Region growing is then utilized based on the two best points. The region growing algorithm is limited by estimated radius, to avoid leakage, and extraction of the aorta, and it is limited by range and step size. This results in a segmentation, and the centerline is extracted by simply tracking the center of the segmentation.

Bauer et al[6, 5] uses a ridge traversal approach based on the Hessian matrix generated from the intensity changes in the original input image. Several of the most promising points from the vesselness algorithm is selected, and for each point the closest ridge is traversed in both directions. The ridge traversal is supported by avoiding sudden shifts in direction and leniency for a limited distance to reduce noise problems. After all the promising points have been traversed the shortest traversals are discarded. The remaining lines are then attempted linked up based on bio-mechanical metrics in a cost function focusing on angle between lines to be connected and distance.

Then all of the methods has a post-processing routine where the longest connected centerline, where the average radius is within some threshhold and the HU value within some threshhold, is selected as the complete centerline for the coronary arteries. All of these methods are very close to each other in results.

In practise it seems like the usage of expert knowledge is a requirement to achieve accurate results when it comes to coronary artery centerline detection and extraction.

2.5.4 Machine Learning

Machine learning is a broad term used to simply describe algorithms that in some way improve, or adapt based on data or training. There are several sub categories of approaches to machine learning. The review by Kirbas et al[25] only covers *Neural network* based machine learning approaches, and the review by Dehkordi et al[15] only mentions it. Yet, two of the algorithms tested on the MICCAI'08[7, 17, 18] framework utilizes machine learning. Their performance is comparable to the other algorithms ranked by the MICCAI'08[7, 17, 18] framework.

Kiramura et al[55] algorithm uses the Hessian matrix combined with machine learning to detect candidate points within the coronary arteries. They describe the machine learning component as a classifier, as its purpose is to determine whether a structure is a part of the vessel tree or not. The classifier is thought by the *Adaptive Boosting* algorithm[20] by exposing it to both positive and negative training samples. The positive training samples are manually labeled centerlines and vessel contours. The adaptive boosting algorithm learns the classifier that discriminates the positive and negative samples through sequentially combining the feature vectors of the samples. The feature vectors where extracted using Haarlike filtering[48]. The candidate points are then connected through a graph reconstruction algorithm consisting of three steps, shape model, matching method and energy matching. The construction of the shape model is done by placing 30 relative connected points. The relative position of these points is decided through the same machine learning algorithm mentioned earlier. Then the extracted candidate points matched to this model by applying an energy function that works in a similar manner to the *Snake* algorithm described in the model-based approach chapter. This ensures that the original candidate points keep their general shape while still adapting to the model shape. This results in a centerline for the coronary arteries.

Schaap et al[41] proposes an algorithm fairly similar to the algorithm presented by Kiramura et al[55]. Schaap et al[41] utilizes machine learning to attempt to establish a general shape of the vessel structures normally observed in the coronary arteries. Their method is coined at improving an already existing centerline, and they describe this as a rough centerline that can either be manually labeled or delivered by some other centerline extraction algorithm. Schaap et al[41] algorithm improves upon the rough centerline in a similar fashion as Kiramura et al[55] where the rough centerline is deformed based on several factors to more closely resemble the general shape of the thought vessel tree.

2.6 Conclusion from background study

For this thesis the objective is to implement an automatic coronary artery segmentation algorithm, and to utilize the parallel capabilities of a GPU. Based on these requirements some of the approaches presented above can be eliminated as options for this thesis. Any of the algorithms presented that are not fully automatic, meaning that they require more then just a single CT image as input are not applicable. Second, it will be advantageous to use an approach that is compliant with a parallel

implementation. While it is possible to utilize a GPU with a serial approach, it is much slower and in many cases both less reliable and slower then a serial implementation for a CPU. Third, while the assignment doesn't specifically say anything about it, choosing an approach that relies on large amounts of training data could also lead to difficulties. As training data for segmentation algorithms normally are manual segmentations performed by experts in the field. These manual segmentations are very time consuming to make, and most approaches that requires training data need many of them.

Therefore, many of the approaches covered in the Neural Network and Artificial Intelligence sections are non applicable for this thesis. Some of the approaches covered in Pattern Recognition, Model-based Tracking and Miscellaneous also relies on additional human interaction, and are also non applicable. The methods presented by Bauer et al[6, 5], Krissian et al[28] and Yang et al[53] all fulfill these requirements. The method presented by Krissian et al[28] utilizes region growing from a low number of seeds. This means that the method is not very applicable for parallel execution. Yang et al[53] and Bauer et al[6, 5] are very similar, where their approaches can easily be divided into stages, and the same stages for both are very applicable for parallel execution. Bauer et al[6, 5] approach is slightly more applicable for a parallel execution. They utilize the Hessian matrix, with the corresponding eigenvectors and eigenvalues, preparing and processing this information is data parallel. Then a *"simple"* vesselness filter is used twice, both of these are also data parallel, before a ridge traversal is performed which largely isn't very applicable for a parallel execution. Yang et al[53] also uses a ridge traversal method that suffers from the same parallel execution issues as Bauer et al[6, 5] approach. Both Yang et al[53] and Bauer et al[6, 5] presents methods for segmentation based on the centerlines they produce. Therefore, the approach we have chosen to focus on is the one by Bauer et al[6, 5], supported by some of the methods presented by Krissian et al[28].

Chapter 3

Methodology

This chapter describes our implementation of a coronary artery segmentation and centerline extraction algorithm. As stated in the conclusion of the background study we have chosen to base our implementation on the method presented by Bauer et al[6, 5]. As stated in the background chapter, our implementation will be in the existing framework *FAST*[43]. This framework was made with the purpose to support parallel implementations on GPUs and works as a solid base for our implementation. FAST utilizes C++ and OpenCL, and those two programming languages will therefore also be the languages we use in our implementation.

The method proposed by Bauer et al[6, 5] presents a collection of steps, ordered in a pipeline. The steps are: Pre-processing, Tubular detection filter, ridge traversal, grouping and linking, and segmentation.

The first step, pre-processing is a collection of operations and algorithms. These algorithms and operations include optional de-noising, creation of the gradient vector field, gradient vector flow, and conversion to Hounsefield units. There are two de-noising filters available in *FAST*[43], Gaussian smoothing and Non-local means. In Bauer et al[6, 5] method the de-noising is stated as optional, so in the results and discussion chapter we will present results with and without these filters applied, and later discuss whether they should be utilized or not. All the components of the pre-processing step will be explained in greater detail in chapter 3.1.

The tubular detection filter step involves running the tubular detection filter twice. The first is run on the gradient vector field, the second is run on gradient vector

field with gradient vector flow. The purpose of doing this is to detect both small and large tubular structures. Bauer et al[6, 5] uses a vesselness filter purposed by Frangi[19]. In our implementation we have also used this, yet in FAST there is an alternative filter implemented, known as a circle fitting tubular detection filter proposed by Krissian et al[28, 27]. This tubular detection filter is more computationally expensive to execute, yet if it provides fewer but more accurate points the overall execution time can be lowered. Therefore our implementation can utilize either of these filters. In the results and discussion chapter we will present results for both of these tubular detection filters, and their corresponding execution times.

The ridge traversal step is closely linked with the linking and grouping step. The ridge traversal algorithm proposed by Bauer et al[6, 5] produces several centerlines. Most of these centerlines should form a graph tree, yet because of the nature of the coronary arteries there will probably be some smaller vessels that are not connected to the overall tree. The grouping and linking step processes those and attempt to connect them to the final centerline tree. Both of these steps are implemented according to the description by Bauer et al[6, 5].

The final step is the segmentation step. The inverse gradient flow tracking algorithm is described by Bauer et al[6, 5] but not utilized for segmentation of the coronary arteries as the overall objective of their proposal was to produce an accurate centerline extraction. Yet the inverse gradient flow tracking algorithm is used here because it utilizes the gradient vector field, and it has been proposed by Bauer et al[6, 5] for similar vessel segmentation purposes.

3.1 Pre-processing

3.1.1 Hounsefield unit conversion

The intensity values of X-ray and CT images are measured in Hounsefield Units (See chapter 2.2.2). These values correspond to the radiation absorption amount of the tissue at a specific location, and can therefore be used to distinguish different types of tissue from each other. Air is estimated to be around -1000 HU, while bone is roughly between 700 and 3000 HU. In this thesis the objective is to extract the centerlines of and segment the coronary arteries from CTA images. As stated earlier in chapter 2.1 and 2.2 a contrast fluid is injected into the vessels in question

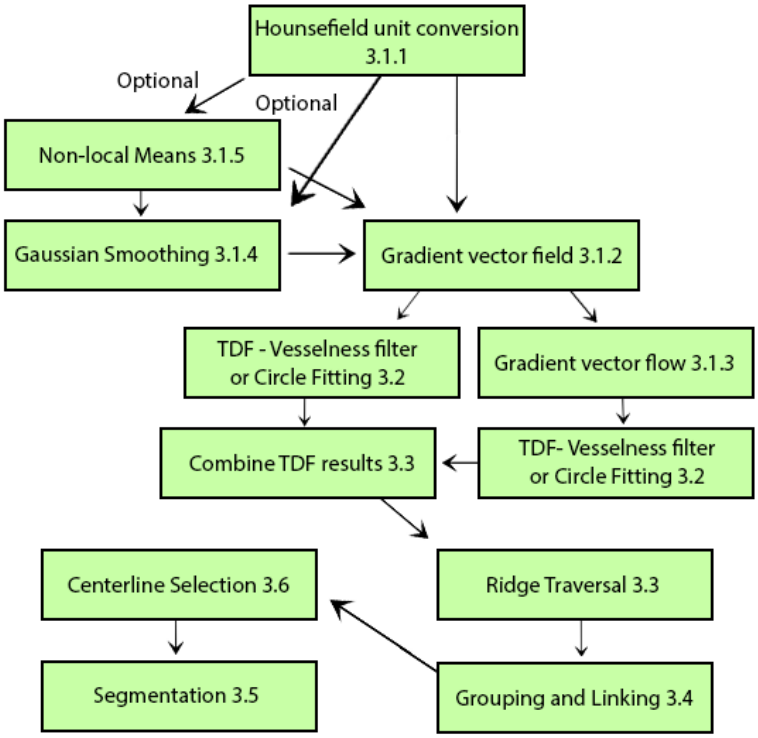

Figure 3.1: Diagram of algorithm

to makes them more visible in the image. Vessels with contrast fluid is estimated to have a HU value around 200.

Any value significantly outside of this range is largely irrelevant for this thesis. In chapter 3.1.2 and 3.1.3 we will see that the gradient results depends on the intensity displayed in the image. Therefore to limit the amount of false positives, and to reduce the amount of data the other parts of the implementation needs to process, the HU values observed in the image is capped at 500. Any value greater than that is simply set to 500. The same action is performed on any value lower then 0, where any lower value is set to 0. These two variables are labeled HU_{min} and HU_{max}. The remaining values between 0 and 500 are scaled so that the range of intensity values is converted to a floating point number from 0.0 to 1.0. The equation used for this can be seen in equation[3.1].

$$I(\vec{v}) = \begin{cases} 1.0 & \text{if} \quad I(\vec{v} \geq HU_{max}) \\ \frac{I(\vec{v}) - HU_{min}}{HU_{max} - HU_{min}} & \text{else} \end{cases} \tag{3.1}$$

3.1.2 Gradient vector field

The purpose of this step is to prepare information for several other steps. Tubular Detection Filters (TDFs) are used to detect tubular structures, such as vessels in 3D images. TDFs usually perform some sort of shape analysis on each voxel in an image and return a value indicating the likelihood that a specific voxel is a part of a tubular structure.

Hessian-based TDFs are TDFs that uses the *Hessian Matrix* for the shape analysis. The Hessian matrix is a set of variables that represent the second-order derivative information at any specific voxel position, while the first order derivative information at any specific voxel in each direction is refereed to as the gradient. The first order derivative information, denoted $\nabla I(\vec{v}) = (\frac{\partial I(\vec{v})}{\partial x}, \frac{\partial I(\vec{v})}{\partial y}, \frac{\partial I(\vec{v})}{\partial z})$, where $\vec{v}$ is a voxel. These vectors corresponds to the change in intensity values in all three direction at the voxel position. The direction of these vectors indicate the direction of largest intensity change, and the magnitude of the vector $|\nabla I(\vec{v})|$ indicates how much the intensity changes in that direction. The second-order derivative information needed

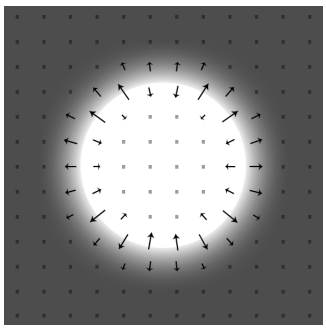

Figure 3.2: Gradients of a sliced, ideal tube

for the Hessian matrix can then be extracted from the first-order derivatives by calculating the gradients again on each component of the first-order derivative. The Hessian matrix $H(\vec{v})$ is a matrix of these three gradients. Each of these gradients describes the change of the change of intensity in their respective direction.

$$H(\vec{v}) = \begin{bmatrix} \nabla(\nabla I(\vec{v})_x) \\ \nabla(\nabla I(\vec{v})_y) \\ \nabla(\nabla I(\vec{v})_z) \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 I(\vec{v})}{\partial xx} & \frac{\partial^2 I(\vec{v})}{\partial xy} & \frac{\partial^2 I(\vec{v})}{\partial xz} \\ \frac{\partial^2 I(\vec{v})}{\partial yx} & \frac{\partial^2 I(\vec{v})}{\partial yy} & \frac{\partial^2 I(\vec{v})}{\partial yz} \\ \frac{\partial^2 I(\vec{v})}{\partial zx} & \frac{\partial^2 I(\vec{v})}{\partial zy} & \frac{\partial^2 I(\vec{v})}{\partial zz} \end{bmatrix} \tag{3.2}$$

Calculating the Hessian matrix for each voxel is needed for both of the TDFs described in chapter 3.2.2 and 3.2.3. The task of calculating the Hessian matrix is very well suited for a parallel execution. Each voxel in the image can be processed individually while only relaying on information from the original image. The product created when all voxels have been processed is refereed to as the gradient vector field in this thesis.

In our implementation we calculate the gradients by using a central difference scheme which looks at two neighbouring voxels in each direction and calculates the variation as shown below (x,y,z indicates direction):

$$\nabla I(\vec{v})_x = \frac{I(\vec{v} + (1,0,0)) - I(\vec{v} - (1,0,0))}{2} \tag{3.3}$$

$$\nabla I(\vec{v})_y = \frac{I(\vec{v} + (0,1,0)) - I(\vec{v} - (0,1,0))}{2} \tag{3.4}$$

$$\nabla I(\vec{v})_z = \frac{I(\vec{v} + (0,0,1)) - I(\vec{v} - (0,0,1))}{2} \tag{3.5}$$

To further prepare the gradients for the next step in pre-processing (gradient vector flow), they need to be normalized. The normalization is required to promote contrast invariance, and to make sure that the gradient information that is significant for this project is maintained through the *gradient vector flow* process. The parameter F_{max} is used as a threshold for the gradient magnitudes, all magnitudes above this value will then be scaled to max length. This is necessary for the next step *gradient vector flow* to work, and it makes the gradient creation and consequently the *gradient vector flow* less sensitive to noise. The equation below shows how the gradient vector field is normalized:

$$\vec{V}^n(\vec{v}) = \begin{cases} \frac{\vec{V}(\vec{v})}{|\vec{V}(\vec{v})|} & \text{if} \quad |\vec{V}(\vec{v})| \geq F_{max} \\ \frac{\vec{V}(\vec{v})}{F_{max}} & \text{else} \end{cases} \tag{3.6}$$

3.1.3 Gradient vector flow

The purpose of gradient vector flow is to propagate the gradient information away from the original gradients. While this to some extent will diffuse the gradients, it will maintain the original information where the gradients is of a *significant* magnitude. This quality, in the context of tubular detection is known as edge preservation or feature preservation. This is because all the Hessian-based TDFs detect tubular structures based on the assumption that the edge of a structure will either be at a *valley* or *ridge*, meaning that the edge will be where two or more gradients point towards each other. In this case, the purpose of using gradient vector flow is to enable gradient information at positions further away from an edge while preserving the original edge information. This is necessary for detection of larger structures, and for the segmentation algorithm described in chapter 3.5. Gradient vector flow

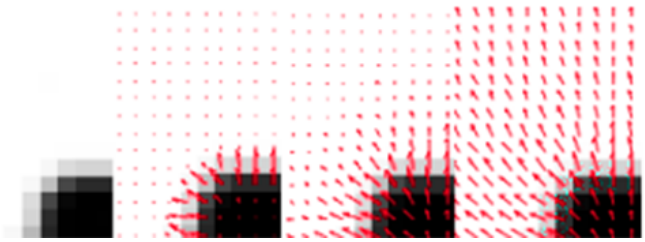

Figure 3.3: Gradient vector flow illustration

was originally introduced by Xu and Prince[52] as an external force field for active contours. The gradient vector field created by the gradient vector flow attempts to minimize the energy function below.

$$E(\vec{V}) = \int \mu |\nabla \vec{V}(\vec{x})|^2 + |\vec{V}_0(\vec{x})|^2 |\vec{V}(\vec{x}) - \vec{V}_0(\vec{x})|^2 d\vec{x} \tag{3.7}$$

Xu and Prince[52] also presents a solution to iteratively utilize gradient vector flow by solving the following Euler equation for each vector component independently:

$$\vec{0} = \mu \nabla^2 \vec{V} - (\vec{V} - \vec{V}_0)|\vec{V}_0|^2 \tag{3.8}$$

This equation is solved by treating $\vec{V}$ as a time function and the resulting equation can be solved by utilizing the numerical scheme shown in algorithm 1. The laplacian $\nabla^2\vec{V}(\vec{v})$ is approximated by using a 6 point finite difference scheme.

Algorithm 1: Gradient Vector Flow

for *implementation specific number of iterations* **do**
  for *all point* $\vec{v} = (x, y, z)$ *in image* **do**
    laplacian $\leftarrow -6\vec{V}(\vec{v}) + \vec{V}(x+1, y, z) + \vec{V}(x-1, y, z) + \vec{V}(x, y+1, z) + \vec{V}(x, y-1, z) + \vec{V}(x, y, z+1) + \vec{V}(x, y, z-1)$;
    $\vec{V}(\vec{v}) \leftarrow \vec{V}(\vec{v}) + \mu *$ laplacian $-(\vec{V}(\vec{v}) - \vec{V}_0(\vec{v}))|\vec{V}_0(\vec{v})|^2$;
  end
end

3.1.4 Gaussian Smoothing

Gaussian Smoothing as a very common algorithm to reduce noise and to reduce detail. It is widely used and has many applications. In this case it can be useful to enhance the propagation of information in Gradient Vector Flow. The purpose of GVF is to propagate the edge information towards the center, so that the Tubular Detection Filters can utilize the information later. This can be done by running many iterations of GVF, or it can be made easier by applying a Gaussian Smoothing filter on the image first to aid the GVF in the propagation.

Gaussian Smoothing is done by convolution of the image with a Gaussian kernel with a standard deviation σ. The Gaussian Smoothing algorithm utilized in our implementation is included in *FAST* and utilizes discrete convolution by calculating a new value for each voxel based on the weighted sum of the neighbouring voxels. The size of the neighbourhood is determined by σ in the following manner, $N = 2[3\sigma] + 1$, giving a NxNxN neighbourhood. The weight for the current voxel, and the each neighbouring voxel is calculated with the equation shown below. W is for normalization, and is equal to the sum of the weights.

$$(I * G_\sigma)(\vec{v}) = \frac{1}{W} \sum_{x=-[3\sigma]}^{[3\sigma]} \sum_{y=-[3\sigma]}^{[3\sigma]} \sum_{z=-[3\sigma]}^{[3\sigma]} I(\vec{v}) e^{-\frac{x^2+y^2+z^2}{2\sigma^2}} \tag{3.9}$$

3.1.5 Non-Local Means

Non-Local Means is an algorithm used to reduce noise. It does this on a per voxel basis, making it ideal to be executed on a GPU. For each voxel the algorithm inspects several neighbourhoods surrounding the voxel in an attempt to identify neighbourhoods that are similar to the neighbourhood surrounding the original voxel. Each neighbourhood inspected is given a weight dependent on how similar the neighbourhood is. This is based on the assumption that the averaging of voxels in a similar environment, with a similar color/intensity will produce the *true* voxel value.

Ideally, the search for voxels with similar environments should be done in the entire image, thus the name *Non-local*. In practise though that would be very computationally expensive, so a limitation, or *window*(W_{size}) is introduced. The *window*(W_{size}) limits the search area around a the voxel to be de-noised. Each voxel inside the window acts as a centerpoint for an environment. These environments are normally refereed to as groups. The *group*(G_{size}) around the voxel to be de-noised, and each group around each voxel in the window, is compared in the search for viable voxels to include in the averaging. There are several variations on how the comparisons between groups are judged and weighted. The two most common is, a genuine average of the voxel value of all the voxels except the center voxel, and a Gaussian kernel. Since Gaussian Smoothing will be utilized in this project the averaging method will be used. In addition to the averaging, a de-noising parameter(NLM_s) is introduced, meant to balance the effect of the algorithm. Psudo-code of our implementation of *Non-Local Means*, for each voxel in the CTA image can be found in Algorithm 2.

3.2 Tubular detection filters

As mentioned earlier, the two TDFs that will be utilized in this thesis are Frangi's vesselness filter and Circle Fitting TDF by Krissian et al[28, 27]. These two represent the two major categories of TDFs, central TDFs and offset TDFs. Central TDFs

Algorithm 2: Non-Local Means

$V_p \leftarrow$ voxel to be de-noised;
for *each voxel* w *within* W_{size} **do**
  for *each voxel* p_i *around* v *within* G_{size} **do**
    $C \leftarrow p_i - V_{p_i}$
  end
  NormSum += $C * NLM_s$;
  Sum += C;
end
$V_p \leftarrow Sum/NormSum$

is a subcategory of TDFs that only use information at or close to the current voxel. While Offset TDFs use information at specific offsets (usually predetermined) from the current voxel, as well as information from the current voxel. Offset TDFs can utilize more information than central TDFs and that can give higher accuracy, but the added complexity increases the execution time.

Both the vesselness filter and the circle fitting TDF are, as mentioned earlier, Hessian-based TDFs. That means that they utilize the Hessian matrix in an attempt to determine whether a voxel is a part of a tubular structure or not. This is based on four basic observations about tubular structures and their corresponding first and second order derivatives.

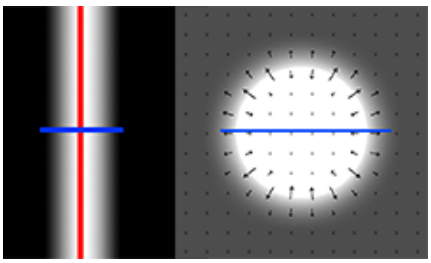

Figure 3.4: Slice views of ideal tubular structures and gradients

The four observation can be deducted from the figures 3.4, 3.5, 3.6.

1. The smallest change in intensity is in the same direction as the direction of the tube. (See figure 3.4)

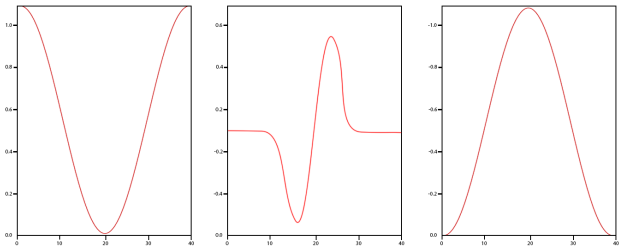

Figure 3.5: Graph of first order derivatives

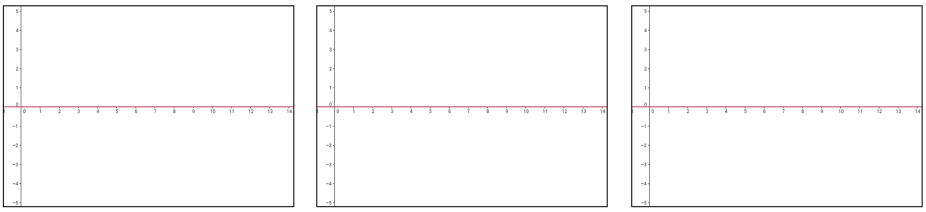

Figure 3.6: Graphcs of second order derivatives

2. The largest change in intensity is in the cross-section plane of the tube (see figure 3.4)

3.The gradient, or first order derivatives, creates a *valley* or *ridge* at the center of the tube depending on whether the tube is dark or light.(see figure 3.4 and 3.5)

4. The largest change in intensity $\partial^2 I(\vec{v})$ is at the center of the tube, because the gradients present there change direction.(see figure 3.5)

These four observations can be used to detect tubular structures. This could be done by checking all possible tube direction and their corresponding derivatives, but this would be a very computationally expensive task. Therefore, an alternative solutions is needed, and the eigenanalysis with it's corresponding eigenvectors and eigenvalues are just that.

3.2.1 Eigenvectors and Eigenvalues

The N eigenvectors of a NxN matrix are non-zero vectors with N components. Eigenvectors have the property that when multiplied with their corresponding matrix, they remain parallel to that matrix. Each eigenvector $\vec{e_i}$ has a corresponding eigenvalue λ_i. That is the factor the eigenvector is scaled by when multiplied with the matrix.

$$H\vec{e}_i = \lambda_i \vec{e}_i \tag{3.10}$$

The Hessian matrix is a 3x3 symmetric matrix, and therefore it has 3 eigenvectors and 3 eigenvalues. The eigenvectors are orthonormal, meaning that they are all normal to each other. The eigenvectors can be interpreted geometrically. The eigenvectors correspond to the directions of the second order derivatives, which are the directions in the volume where the curvature is both the largest and smallest. Based on observation 1. and 2. we can determine that one of the three eigenvectors will correspond to the direction of the tube, and that the other two will be in the cross-sectional plane of the tube. To determine which eigenvector is which, we can use the eigenvalues.

If we sort the eigenvalues, and their corresponding eigenvectors so that we have the relation: $|\lambda_1| < |\lambda_2| < |\lambda_3|$, the direction of the tube will be given by $\vec{e}_1$ because it is the eigenvector with the smallest magnitude. The eigenvalues correspond to the principal curvature (in this case change in intensity change) which means that they correlate to the amount of curvature. Based on observation 1. we can conclude that the eigenvector with the smallest eigenvalue magnitude will also point in the direction of the tube. Based on this we can also determine that $\vec{e}_2$ and $\vec{e}_3$ will be in the cross-sectional plane of the tube and therefore have high eigenvalues. As mentioned earlier, this is because the change in intensity is higher in the cross-sectional plane of the tube, and because all the eigenvectors are orthonormal. The figure below illustrates how the eigenvectors relate to each other and to a tube. We use the

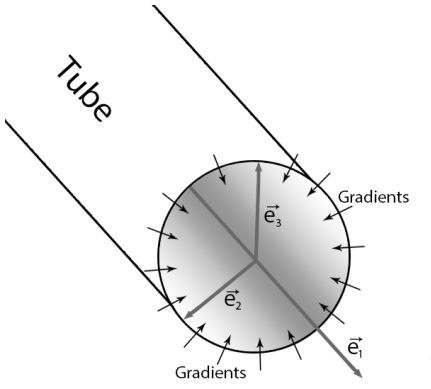

Figure 3.7: Ideal tubular structure with eigenvectors

absolute value of the eigenvalues because the sign of the eigenvalues only indicate the direction of the gradient. While this is relevant in specific cases, it only determines if the tube is white or black, and the background is white or black. If the tube is white and the background is black, λ_2 and λ_3 will be negative. In the reverse case, both λ_2 and λ_3 will be positive. Ideally, based on the figure, and the observations mentioned earlier, a tubular structure with its corresponding eigenvectors and eigenvalues should have these qualities:

$$|\lambda_1| \approx 0 \tag{3.11}$$

$$|\lambda_1| << |\lambda_2| \tag{3.12}$$

$$|\lambda_2| \approx |\lambda_3| \tag{3.13}$$

In table 3.1 we show which type of structures correspond to different configurations of eigenvalues. The Hessian matrix and the eigenanalysis to detect tubular

λ_1	λ_2	λ_3	Structure
-H	-H	-H	Blob (Dark)
+H	+H	+H	Blob (Dark)
L	+H	+H	Tubular (Dark)
L	-H	-H	Tubular (Bright)
L	L	+H	Plate (Dark)
L	L	-H	Plate (Bright)

Table 3.1: Table of tubular shapes

structures relies on the gradient information. In the event where there is very little intensity change, like in the center of a large tubular structure, it is not possible to calculate the Hessian. Therefore a technique for propagating the needed information away from the edges is needed. In this case that method is gradient vector flow and is described in 3.1.3.

There are several algorithms for calculating eigenvalues and eigenvectors of a matrix. The one used in *FAST* is QL, The idea of QL is that, that any real matrix can be decomposed to the form:

$$H = Q \cdot L \tag{3.14}$$

Where Q is the orthogonal matrix and L is the lower triangle of the Hessian. The Householder transformation is used to get this decomposition. QL is an iterative algorithm that performs a sequence of transformations that will eventually converge into the eigenvalues and the eigenvectors. QL start with $H_0 = H$, then for each iteration i it finds the orthogonal and lower triangle matrix of the current matrix H_i and generate the next matrix H_{i+1} by applying the equation below.

$$H_{i+1} = L_i \cdot Q_i \tag{3.15}$$

After several iterations the eigenvectors will appear at the columns of the orthogonal matrix Q_i and the eigenvalues will be on the diagonal of the L_i matrix. The time complexity for QL is $O(n^3)$ per iteration. The implementation in *FAST* is an implicit adaptation from the tql2 subroutine from the Fortran library EISPACK.

3.2.2 Frangi vesselness filter

The Frangi vesselness filter[19] is a very popular, and fairly simple TDF. It uses three variables: a, b and s, in addition to the eigenvalues. These three variables are used for weighting of three criteria, R_a, R_b, and S. R_a and R_b are geometric criteria, meaning that they attempt to determine what shape that is present at and around the current voxel, S is a noise handling criteria. Each of these criteria are weighted by the three variables, this gives the filter flexibility, and adaptability to handle varying circumstances.

R_a purpose is to distinguish between plate-like and line-like structures, while R_b indicates blob-like structure. And the purpose of S as mentioned earlier, is to provide robustness against noise.

$$R_a = \frac{|\lambda_2|}{|\lambda_3|} \tag{3.16}$$

$$R_b = \frac{|\lambda_1|}{\sqrt{|\lambda_2||\lambda_3|}} \tag{3.17}$$

$$S = \sqrt{|\lambda_1|^2 + |\lambda_2|^2 + |\lambda_3|^2} \tag{3.18}$$

These criteria makes more sense if you consider the eigenvalues as the length of each of an ellipsoids axis. This also makes it clearer that the criteria above relies on the eigenvalues being sorted to form the relation $|\lambda_1| < |\lambda_2| < |\lambda_3|$. By imagining this, a perfect ellipsoid in this case would be almost flat in that $|\lambda_1|$ should be very small, and $|\lambda_2|$ and $|\lambda_3|$ should be large. Frangi et al[19] combines R_a, R_b and S into the expression below, where a tube-like structure should have a high R_a and a low R_b.

$$V = (1 - exp\Big(-\frac{R_a^2}{2a^2}\Big))exp\Big(-\frac{R_b^2}{2b^2}\Big)(1 - exp\Big(-\frac{S^2}{2c^2}\Big)) \tag{3.19}$$

To detect both small and large tubular structures the Frangi vesselness filter[19] is run twice. First it is applied to the gradients produced in pre-processing (chapter 3.1.2), then it is applied to the resulting gradients after the original gradients have been treated with the gradient vector flow algorithm (chapter 3.1.3). This produces two outputs for every voxel in the original image, and they will both be utilized later by the ridge traversal algorithm (chapter 3.3).

3.2.3 Circle-fitting TDF

The Circle Fitting TDF by Krissian et al[28, 27] is a computationally more expensive method than the vesselness filter, yet it can return better results. It works by using each voxel as a center point, and from there slowly expand a circle until some border is reached. As stated earlier, an ideal circular structure should have a very small $|\lambda_1|$ value, and larger $|\lambda_2|$ and $|\lambda_3|$ values. The Circle Fitting algorithm does not utilize the eigenvalues, but rather the eigenvectors, and the same rule applies for them.

The two eigenvectors $\vec{e}_2$ and $\vec{e}_3$ is used to construct a circle in the cross-sectional plane. First a very small radius is used, defined by a $minRadius$ parameter. Then, for a N number of positions along the circle the radius r will be sampled. The individual direction to each position $\vec{d}_i$ is calculated in the following way:

$$\vec{d}_i = \vec{e}_2 \, sin\frac{2\pi i}{N} + \vec{e}_3 \, cos\frac{2\pi i}{N} \tag{3.20}$$

The equation above, with the center point and the current radius makes up the individual score for each point along the circle. The actual TDF value, is the largest average dot product between all the sampled points. The average will continue to increase as long as the gradients continue to increase in length. When this does not occur, the circle has met the border, and the largest previous value calculated will be the TDF output. This requires a step-wise approach to at what radius each average is calculated at, in the implementation used in this thesis that step is 0.5. The circle expansion is also limited by how large it is allowed to grow ($maxRadius$), this is to ensure that vessel structures outside of the range we are looking for are avoided in in the ridge traversal algorithm. As mentioned earlier, the average dot product between all the sampled points, as well as their inward normal, makes up the TDF score for the current radius. The complete formula to calculate the average for each radius achieved is below:

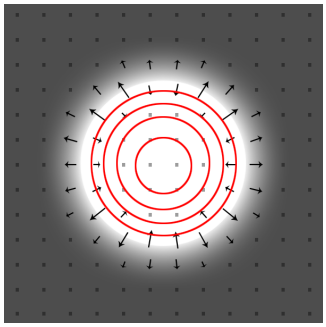

Figure 3.8: Circle Fitting illustration

$$T(\vec{v}, r, N) = \frac{1}{N} \sum_{i=0}^{N-1} \vec{V}(\vec{v} + r\vec{d_i}) \cdot (-\vec{d_i}) \tag{3.21}$$

In *FAST* this is implemented to be calculated on a GPU, where each voxel position is processed individually. Below is pseudo code for the implementation in *FAST*, given voxel $\vec{v}$:

Algorithm 3: Circle Fitting TDF

```
Calculate e1 , e2 , e3 at position v;
maxSum ← 0;
for r from minRadius to maxRadius do
    sum ← 0;
    for i from 0 to N-1 do
        d_i ← e2 sin(2πi/N) + e3 cos(2πi/N);
        sum ← sum + V(v r d_i) · (−d_i);
    end
    if sum > maxSum then
        maxSum ← sum;
    else
        break;
    end
end
return maxSum
```

3.3 Ridge traversal

While the TDF output could be used as a centerline, deciding where to place the threshold for which TDF responses to include, and sorting away responses that are not relevant for the current problem is difficult. Both TDFs also return unreliable results for vessel junctions and vessels with irregular shape, that can to some extend be counteracted by a ridge traversal algorithm. Using a ridge traversal algorithm is possible when the TDF have a medialness property. Medialness scribes how ”in the center” a point is of a tubular structure. Both Frangi's vesselness filter[19] and Krissians Circle Fitting TDF[27] has this property. The largest TDF responses, from both TDFs will be in the center of the structure in question, as points close to any of the edges will have a lower TDF output score.

The ridge traversal algorithm presented by Bauer et al[6, 5] involves two steps. The first step is to locate valid starting points for the traversal. This is done by considering all the TDF outputs (from both TDF runs) and choosing the points that are both over a given threshold ($tHigh$) and have the highest TDF value among it's 26 neighbours. The points that qualify are placed into a list, sorted with focus on TDF score.

When all the possible TDF points has been considered, the ridge traversal algorithm takes the top point (the point with the current highest TDF score) and uses it as a start point for traversal. From this start point x_0 we use the lowest eigenvector $\vec{e}_1$ to determine the general direction of the tube. Then, starting from x_0 the traversal is performed independently for both directions t_0 and $-t_0$. The next point in the traversal is chosen based on the direction from the current point to the candidate point, and the TDF score of the candidate point. The directional restrictions are calculated based on a threshold to avoid large angle changes in this way: $\vec{x_i x_i^n} \cdot t_i > 0$. All the candidate points that fulfil the directional requirement are considered, and the one with the largest TDF value is chosen. The direction is then updated based on this new point $t_{i+1} = sign(\vec{x_i x_{i+1}} \cdot t_{i+1}) t_{i+1}$, to maintain the direction for the next step in the traversal. This is repeated until one of the following events occur: the TDF values of all the candidate points are below a given threshold ($tLow$), the traversal hits an existing centerline or the traversal hits its own centerline and creates a loop.

In the event that the traversal terminates because it hit an existing centerline, the current centerline is included in the centerline that was hit, and the ridge traversal starts over with the next point from the priority list. Below is pseudo code for the candidate point selection (algorithm 4) and for the ridge traversal algorithm (algorithm 5).

Algorithm 4: Candidate point selection

```
input: TDF output T(x) with directions t(x);
input: threshold tHigh;
queue M;
for each voxel x in T do
    if T(x) > tHigh and T(x) ≥ max_{y∈Adj26(x)} T(y) then
        M.add(x);
    end
end
```

3.4 Grouping and linking

Grouping and linking is an additional method presented by Bauer et al[6, 5]. The purpose of this addition is to counteract some of the common problems with TDFs and ridge traversal. There may be brakes in the centerlines produced by the ridge traversal because of noise, or vessel junctions, or irregular vessel shapes. The grouping and linking approach presented by Bauer et al[6, 5] is a bio-mechanical geometric approach, where each centerline's endpoints are considered. Endpoints are the final point added to the traversal in both directions. The endpoint of a centerline that terminated because it hit an already existing centerline, or it self, is not considered for grouping and linking. The grouping and linking approach presented by Bauer et al[6, 5] is part of a larger, more general approach, but for coronary arteries specifically some parts of the general approach is not used.

The method works by identifying and selecting the endpoint of single centerlines. For each endpoint x_s a search outward in the tangent-direction t_s of the centerline is performed for potential connections to other centerlines. If a candidate point is

Algorithm 5: Ridge Traversal

input: TDF output T(x) with directions t(x);
input: threshold $tLow$;
input: queue M;

set CL ← () //Set of centerlines;
image $C(x) \leftarrow false$ //image indicating centerline positions;

while $M \neq ()$ **do**
  $x_0 \leftarrow$ pop(M);
  if $C(x_0) = false$ **then**
    list cl //centerline for point x_0;
    point x ← x_0;
    direction t ← $t(x_0)$;
    while $T(x) > tLow$ *and* $C(x) = False$ **do**
      cl ← pushFront(cl,x);
      C(x) ← True;
      point $x_n = argmax_{y \in Adj26(x)} T(y) sign(\vec{xy} \cdot t)$;
      direction $t_n = t(x_n) sign(\vec{xx_n} \cdot t(x_n))$;
      $x \leftarrow x_n$;
      $t \leftarrow t_n$;
    end
    cl ← popBack(cl) //to avoid duplicates of x_0;
    point x ← x_0;
    direction t ← $-t(x_0)$;
    while $T(x) > tLow$ *and* $C(x) = False$ **do**
      cl ← pushBack(cl,x);
      C(x) ← True;
      point $x_n = argmax_{y \in Adj26(x)} T(y) sign(\vec{xy} \cdot t)$;
      direction $t_n = t(x_n) sign(\vec{xx_n} \cdot t(x_n))$;
      $x \leftarrow x_n$;
      $t \leftarrow t_n$;
    end
    CL.add(cl);
  end
end
return *CL*

found within some distance limit ($dHigh$), then a connection cost calculation is performed. If the difference in gray value (from the original input image) between the candidate point and the endpoint is to high $|I(x_s) - I(x_e)| > gMax$ the candidate is rejected. If the gray value difference is within the limit a connection cost $C(x_s, t_s, x_e)$ is calculated.

The connection cost is meant to generate a cost that represent a trade of between distance between the two points, x_S and x_e and the angle between the existing centerline and the target centerline as seen in the function below.

$$C(x_s, t_s, x_e) = \frac{||x_s - x_e||}{exp(-\angle(\frac{\vec{x_s}\vec{x_e}, t_s}{2p^2}))} \tag{3.22}$$

An opening angle variable p is introduced to allow for a variable input, in this case p is set to 0.30. Potential connections with too large cost $cMax$ are discarded. Finally, if a candidate has fulfilled all the requirements, a shortest path traversal is performed between the endpoint and the candidate point, and the now connected centerline is added to the candidate centerline.

3.5 Inverse gradient flow tracking

Inverse gradient flow tracking was not used by Bauer et al[6, 5] in correlation with their work on coronary arteries. Yet it was used by them in other similar projects (segmentations of vessels in the liver,brain etc). The inverse gradient flow tracking algorithm presented by Bauer et al[6, 5] is also optimised to utilize both the centerlines produced by the ridge traversal, and the gradient vectors produced by the gradient vector flow algorithm. It works in an serial manner, and is therefore implemented to use a CPU in *FAST*.

The inverse gradient segmentation algorithm works by growing the segmentation from the centerline produced by the previous steps. It does this by growing from the centerline in the inverse direction of the gradients produced by the gradient vector flow. As long as the length of the next gradient vector is larger than the previous length it will keep growing. As mentioned earlier, the change in intensity should be largest at the edge of the tubular structure, therefore the gradient magnitude at the edge should also be at its largest point at the edge.

To make the algorithm more robust, and to make up for potential errors produced in the previous steps, the segmentation algorithm dilate the centerline slightly before

the actual segmentation is performed. This is to avoid false termination, if the centerline is not exactly in the center. Then, each voxel in the centerline is added to a queue, and processed individually. If a neighbouring voxel is not already a part of the segmentation, and if the gradient magnitude is larger then the previous (current) voxel, then it is added to the segmentation. Below is pseudo code for the inverse gradient flow segmentation algorithm.

Algorithm 6: Inverse Gradient Flow Segmentation

```
input: dilated centerline C;
input: gradient field V⃗;
queue Q ← C;
set S ← C while Q ≠ () do
    x⃗ ← Q.pop;
    for each voxel y⃗ ∈ Adj(x⃗) do
        if y⃗ ∉ S and |V⃗(y⃗)| and argmax_{z⃗∈Adj26(y⃗)} ((z⃗−y⃗)·V⃗(y⃗)) / (|(z⃗−y⃗)||V⃗(y⃗)|) == x⃗ then
            S.add(x⃗);
            Q.push(y⃗);
        end
    end
end
return S;
```

3.6 Avoidance of false positives

There are multiple vessels and tubular structures in a CTA of the chest region. Bauer et al[6, 5] discusses a few methods to remove some of the vessels that are not a part of the coronary arteries. Some of the false positives are removed by the thresholding described earlier (chapter 3.1.1). Bauer et al[6, 5] then selects the two largest, connected centerlines. The two largest centerlines with the thresholding is in most cases the coronary arteries, but the method is vulnerable to excluding some of the coronary arteries in the event that some of the vessels are not connected.

3.7 Parameters used

Bauer et al[6, 5] presents the parameters they used for testing. Some of those are compatible with our implementation here, while others are not. The reason that some of the parameters are not compatible is processing time optimization. Our Hounsfield units are scaled from 0 to 1, which is not the case in for Bauer et al[6, 5] implementation. The scaling of the Hounsfield units makes a significant difference, as the processing of normalized floats is very quick on a GPU. We have also introduced a few additional parameters, like the **searchDistance**, this is to limit the range the grouping and linking part of our implementation can search in. This range is to make sure that the algorithm does not search to many unnecessary points, as the candidate requirements will prohibit all the points outside a relative range.

	Parameters used in Pre-processing	
Symbol	Description	Value
HU_{max}	Maximum HU value	500
HU_{min}	Minimum HU value	0
G_{max}	Gradient scaling parameter	0.2
μ	GVF regularization value	0.05
G_i	GVF iterations	500

Table 3.2: Table of Pre-processing Parameters

	De-noising when used	
Symbol	Description	Value
σ	Standard deviation of *Gaussian Smoothing*	0, 0.4, 1.2
W_{size}	Window *radius* of *Non-Local Means*	24
G_{size}	Group *radius* of *Non-Local Means*	3
NLM_s	Strength of *Non-Local Means*	0.3

Table 3.3: Table of Noise reduction Parameters

	Vesselness filter and Circle Fitting TDF	
Symbol	Description	Value
a	R_a weight of Vesselness Filter	0.5
b	R_b weight of Vesselness Filter	0.5
c	S weight of Vesselness Filter	100
R_{min}	Starting radius for Circle Fitting	0.5
R_{max}	Max radius for Circle Fitting	5
R_{step}	Radius growth in Circle Fitting	0.5

Table 3.4: Table of TDF parameters

	Ridge Traversal and Grouping and Linking	
Symbol	Description	Value
D_{min}	Min centerline length	10
T_{high}	Threshold for candidate	0.8
T_{low}	Threshold for low TDF value in centerline	0.3
C_{max}	Connection max for Linking	20
G_{max}	Grayvalue difference max for linking candidate	50
D_{high}	Max distance to search for linking	20
p	Angle restriction for linking	0.3

Table 3.5: Table of Ridge Traversal and Grouping and Linking Parameters

Chapter 4

Results

For testing and verification of our method St.Olavs Hospital and SINTEF Medical Technology provided two anonymized CT data-sets of the heart. In addition to these two, the *Rotterdam Coronary Artery Algorithm Evaluation Framework*(see chapter 4.1) was utilized for verification of the centerline extraction. The verification results will be included for each of the methods tested below, and the produced images of each method will be from the two anonymized CT data-sets. The segmentations produced is illustrated by the *Marching Cubes*[50] algorithm included in *FAST* and OpenGL.

The testing was done on a system with an Intel i5-4460 (3.2GHz) processor, with an NVIDIA Geforce GTX 970 GPU.

4.1 Rotterdam Coronary Artery Algorithm Evaluation Framework

The purpose of the *Rotterdam Coronary Artery Algorithm Evaluation Framework*[7, 17, 18] is to provide a quantitative evaluation and comparison of methods for coronary artery centerline extraction algorithms. It comes with two sets of data, a training set and a testing set. The training data-set consists of 8 CTA images where the manually extracted centerlines are included. The testing data-set consists of 24 CTA images where the manual centerlines are undisclosed.

The evaluation frameworks measures the performance of an algorithm by three percentage grades:

OF: Overlap until first error, measures the continuous centerline length produced without producing a centerline voxel outside the error range.
OT: Overlap with clinically relevant parts of the vessels. Vessel parts with a diameter above 1.5mm.
OV: Overall overlap within the error range.

These measures operate with an error range, where a centerline point doesn't have to be at exactly the same position as the reference centerline. A centerline point is given a graded score based on how far away the point is, where a centerline point is better the closer it is to the centerline point reference. The average distance is reflected in a forth performance measure ***AI***, which gives an average distance in millimeters between the centerlines produced, and the reference centerlines.

4.2 Run-times

Dataset	Size	Runtime
Dataset 1	424x412x396	10min 48sec
Dataset 2	416x456x412	11min 34sec

Table 4.1: Table of total Run-times

Component	Time average
I/O	4sec
Non-local Means	5sec
Gaussian	1sec
Gradient vector field	$<$1sec
Gradient vector flow	8min 31sec
Vesselness Filter	4sec
Circle Fitting	7sec
Ridge Traversal	1min 3sec
Grouping and Linking	1min 9sec
Segmentation	6sec

Table 4.2: Table of Component Run-times

4.3 Centerline extraction and Segmentation

In this section the centerline extraction results produced by the methods described in chapter 3 will be presented. Both the results produced by *Frangi's Vesselness Filter* and *Circle Fitting*, combined with *Non-Local Means* and *Gaussian smoothing* denoising is presented here.

Case	TDF	GS-L	GS-H	NLM	OV	OF	OT	AI
1	Central	No	No	No	86.3%	64.4%	90.1%	0.6
2	Central	Yes	No	No	87.4%	65.0%	90.2%	0.6
3	Central	No	Yes	No	89.7%	64.1%	92.9%	0.5
4	Central	No	No	Yes	86.0%	64.8%	91.3%	0.6
5	Central	Yes	No	Yes	87.6%	63.9%	90.4%	0.5
6	Central	No	Yes	Yes	85.9%	65.3%	87.7%	0.6

Table 4.3: Table of testing results with Vesselness Filter

Case	TDF	GS-L	GS-H	NLM	OV	OF	OT	AI
1	Circle	No	No	No	63.1%	31.9%	70.0%	0.8
2	Circle	Yes	No	No	62.5%	29.4%	66.9%	0.7
3	Circle	No	Yes	No	61.1%	27.9%	66.3%	0.8
4	Circle	No	No	Yes	63.3%	33.2%	70.2%	0.7
5	Circle	Yes	No	Yes	62.3%	32.0%	69.1%	0.8
6	Circle	No	Yes	Yes	60.1%	29.9%	67.0%	0.8

Table 4.4: Table of testing results with Circle Fitting TDF

Figure 4.1: Centerline extraction with Vesselness Filter, no-denoising, dataset 1

Figure 4.2: Centerline extraction with Vesselness Filter, no-denoising dataset 2

Figure 4.3: Segmentation with Vesselness Filter, no-denoising dataset 1

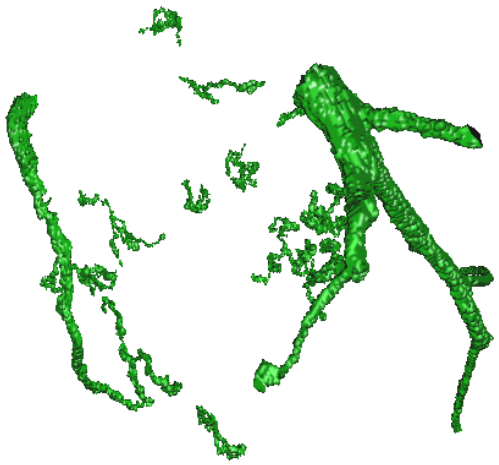

Figure 4.4: Segmentation with Vesselness Filter, no-denoising dataset 2

Figure 4.5: Centerline extraction with Circle Fitting, no-denoising, dataset 1

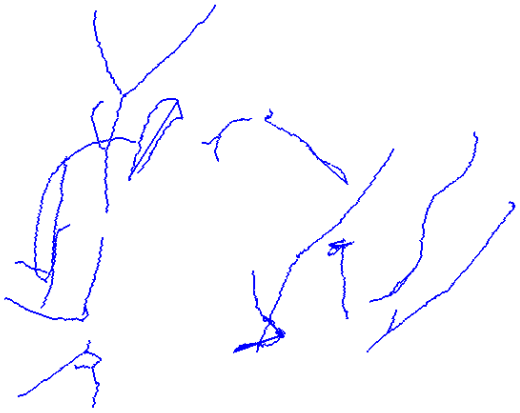

Figure 4.6: Centerline extraction with Circle Fitting, no-denoising dataset 2

Figure 4.7: Segmentation with Circle Fitting, no-denoising dataset 1

Figure 4.8: Segmentation with Circle Fitting, no-denoising dataset 2

Figure 4.9: Centerline extraction with Vesselness Filter, Gaussian Smoothing Low Parameters dataset 1

Figure 4.10: Centerline extraction with Vesselness Filter, Gaussian Smoothing Low Parameters dataset 2

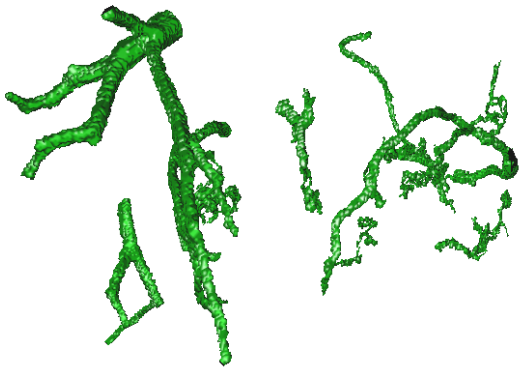

Figure 4.11: Segmentation with Vesselness Filter, Gaussian Smoothing Low Parameters dataset 1

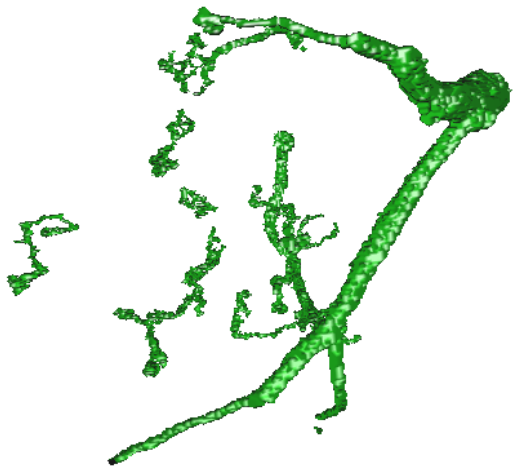

Figure 4.12: Segmentation with Vesselness Filter, Gaussian Smoothing Low Parameters dataset 2

Figure 4.13: Centerline extraction with Circle Fitting, Gaussian Smoothing Low Parameters dataset 1

Figure 4.14: Centerline extraction with Circle Fitting, Gaussian Smoothing Low Parameters dataset 2

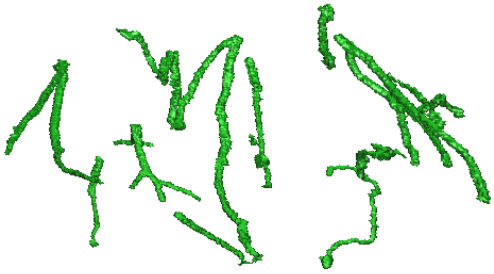

Figure 4.15: Segmentation with Circle Fitting, Gaussian Smoothing Low Parameters dataset 1

Figure 4.16: Segmentation with Circle Fitting, Gaussian Smoothing Low Parameters dataset 2

Figure 4.17: Centerline extraction with Vesselness Filter, Gaussian Smoothing High Parameters dataset 1

Figure 4.18: Centerline extraction with Vesselness Filter, Gaussian Smoothing High Parameters dataset 2

Figure 4.19: Segmentation with Vesselness Filter, Gaussian Smoothing High Parameters dataset 1

Figure 4.20: Segmentation with Vesselness Filter, Gaussian Smoothing High Parameters dataset 2

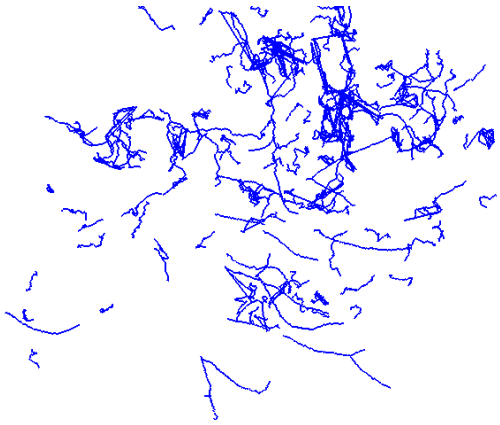

Figure 4.21: Centerline extraction with Circle Fitting, Gaussian Smoothing High Parameters dataset 1

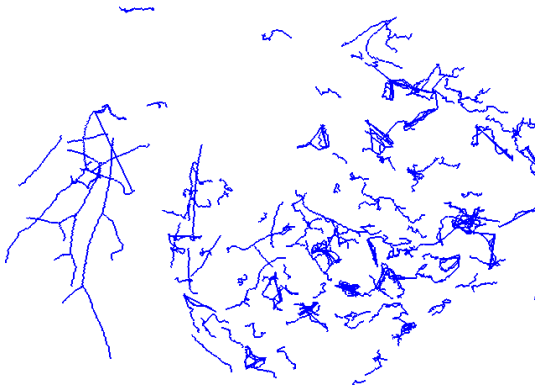

Figure 4.22: Centerline extraction with Circle Fitting, Gaussian Smoothing High Parameters dataset 2

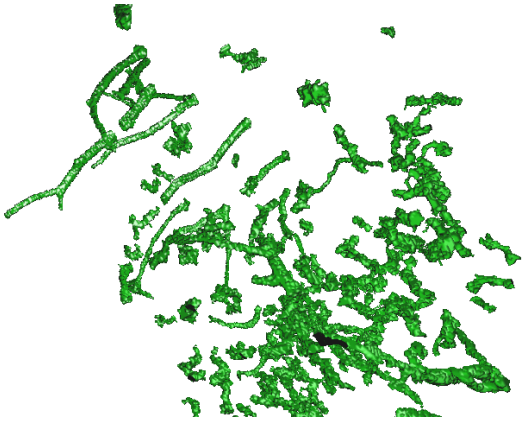

Figure 4.23: Segmentation with Circle Fitting, Gaussian Smoothing High Parameters dataset 1

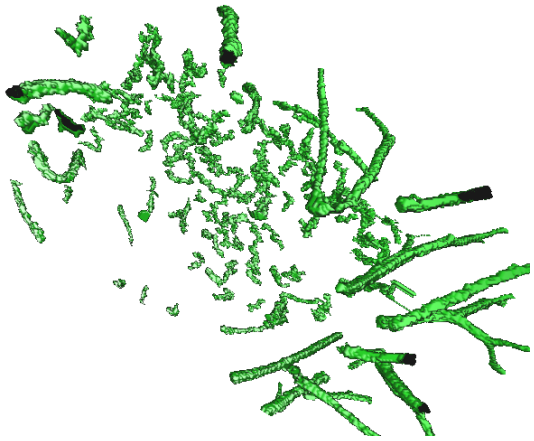

Figure 4.24: Segmentation with Circle Fitting, Gaussian Smoothing High Parameters dataset 2

Figure 4.25: Centerline extraction with Vesselness Filter, Gaussian Smoothing Low Parameters and Non-local Means dataset 1

Figure 4.26: Centerline extraction with Vesselness Filter, Gaussian Smoothing Low Parameters and Non-local Means dataset 2

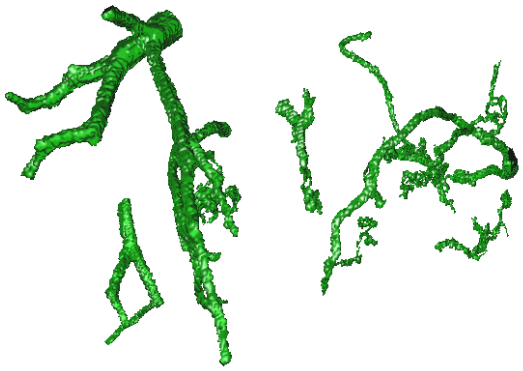

Figure 4.27: Segmentation with Vesselness Filter, Gaussian Smoothing Low Parameters and Non-local Means dataset 1

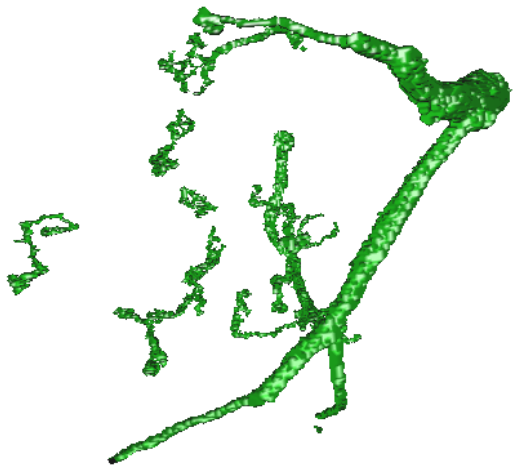

Figure 4.28: Segmentation with Vesselness Filter, Gaussian Smoothing Low Parameters and Non-local Means dataset 2

Figure 4.29: Centerline extraction with Circle Fitting, Gaussian Smoothing Low Parameters and Non-local Means dataset 1

Figure 4.30: Centerline extraction with Circle Fitting, Gaussian Smoothing Low Parameters and Non-local Means dataset 2

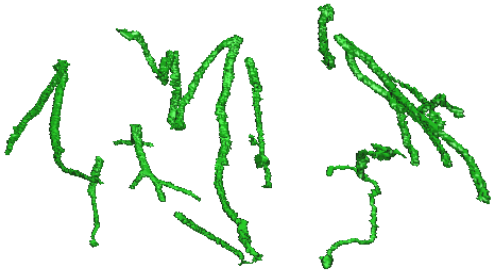

Figure 4.31: Segmentation with Circle Fitting, Gaussian Smoothing Low Parameters and Non-local Means dataset 1

Figure 4.32: Segmentation with Circle Fitting, Gaussian Smoothing Low Parameters and Non-local Means dataset 2

Figure 4.33: Centerline extraction with Vesselness Filter, Gaussian Smoothing High Parameters and Non-local Means dataset 1

Figure 4.34: Centerline extraction with Vesselness Filter, Gaussian Smoothing High Parameters and Non-local Means dataset 2

Figure 4.35: Segmentation with Vesselness Filter, Gaussian Smoothing High Parameters and Non-local Means dataset 1

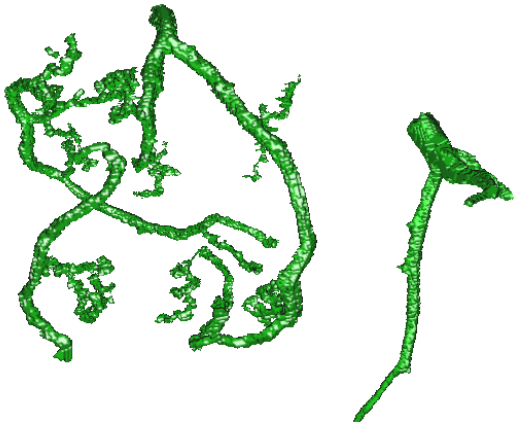

Figure 4.36: Segmentation with Vesselness Filter, Gaussian Smoothing High Parameters and Non-local Means dataset 2

Figure 4.37: Centerline extraction with Circle Fitting, Gaussian Smoothing High Parameters and Non-local Means dataset 1

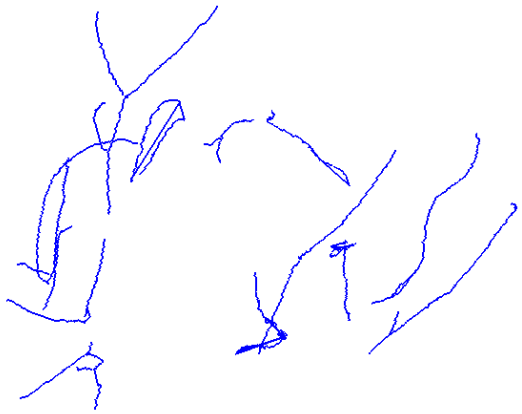

Figure 4.38: Centerline extraction with Circle Fitting, Gaussian Smoothing High Parameters and Non-local Means dataset 2

Figure 4.39: Segmentation with Circle Fitting, Gaussian Smoothing High Parameters and Non-local Means dataset 1

Figure 4.40: Segmentation with Circle Fitting, Gaussian Smoothing High Parameters and Non-local Means dataset 2

Chapter 5

Discussion

In this chapter we will discuss the produced results in chapter 4, with focus on the two TDFs, the effect of the noise filters, the produced centerlines and segmentations and the overall processing time.

5.1 Rotterdam Coronary Artery Algorithm Evaluation Framework

While this framework has been very beneficial and useful for verification of our implementation, it does have some downsides. It performs well in matching, detecting and grading of the centerline points that are relatively close to the reference centerline. But the framework, and the grades it produces does not intuitively provide a grade for how much of the produced centerline is wrong. If every single vessel and tubular structure in the CTA was detected and included in the produced centerline, the grades for both OT and OV would be very good. This is an obvious weakness with the testing framework, as it means that additional, wrong or false centerlines are not very punishing for the grades achieved.

The method that produced the best results was a segmentation and centerline extraction run with the Vesselness filter, and Gaussian smoothing with a σ of 1.2. The results produced by this method in the testing framework are slightly lower to the results reported by Bauer et al[6, 5]. The reason for this could be inaccuracy in the selection of points, as the AI is consistently slightly worse than the results reported by Bauer et al[5, 6].

5.2 Tube Detection Filters

5.2.1 Circle Fitting

The Circle Fitting TDF performs fairly well in terms of run-time, only slightly longer then the less complex Vesselness Filter. It seems like it has no problems detecting larger parts of the coronary arteries, but the smaller sections (<3mm) goes undetected, or is only sporadically detected. This seems to be because of the nature of the Circle Fitting TDF, as the smaller vessels often has less tubular shapes than the larger vessels in the CTA datasets. As explained in chapter 3.2 and 3.2.3 the Circle Fitting TDF expands the radius of a circle stepwise. This means that if the tubular structure is not perfectly circular, the TDF will terminate before the actual edge is reached.

5.2.2 Vesselness Filter

The Vesselness Filter performs well in terms of run-time. It seems to have no problems detecting both small and large tubular structures. Compared to the Circle Fitting TDF it produces both better and more consistent results. One down side of the Vesselness Filter is that it produces a large amount of false positives. If the incorrect centerlines produced are not sorted out before the segmentation, many of the vessels in the lungs will be included. The amount of candidate points produced can also be very high, so the threshold for candidate points for the ridge traversal needs to be set high.

5.3 Noise Filters

5.3.1 Gaussian Smoothing

The Gaussian Smoothing filter performs very well in terms of run-time. It's effect on the extracted centerlines seems largely positive when a high σ is used, but less so with a lower σ. This could indicate that more iterations of the GVF could be used to achieve similar results. It is also a risk to use to much smoothing, as the Gaussian smoothing is not feature preserving, and can hide important image information. Mild smoothing (low σ) also has a minor positive impact on the grades from the testing framework.

Gaussian Smoothing can also to a degree be used to aid the GVF algorithm. Bauer

et al[6, 5] recommends 500 iterations of GVF. While this produces good results, it also takes a long time to execute. With the presence of Gaussian Smoothing, the amount of iterations required to achieve solid results seem to diminish.

5.3.2 Non-Local Means

The Non-Local Means algorithm is fairly slow. It does not have a positive impact on either of the TDFs performance. This result is somewhat surprising. Alone the effect of the NLM noise reduction seems to be negative. If both NLM and Gaussian is run it seems to have a very minor impact on the results. But this minor impact in the testing framework might be nullified in newer and more accurate CTA images.

5.4 Extracted Centerlines

5.4.1 Ridge Traversal

The ridge traversal algorithm is very dependent on the performance of the TDFs. If the TDF produce many and accurate centerline candidates, only a small amount of them has to be utilized as candidate points for the actual tracking. A definite weakness with the traversal is its reliance on long, continuous centerlines. This could be improved with some other selection method for detecting which centerlines belong in the coronary arteries and which centerlines that are false positives (more on this in chapter 6.2).

5.4.2 Grouping and Linking

The grouping and linking approach can be very beneficial to deal with gaps in the centerlines produced by the ridge traversal, but it can also produce false positives. Since the grouping and linking is based on direction and distance between centerline endpoints alone, some of the linking performed could connect vessels that are not actually connected. Detecting these false positives can be very challenging, as they usually occur with very small vessels (< 1mm) that are only partially detected. The testing framework will not detect these either for the reasons stated in chapter 5.1.

5.5 Segmentation Results

The quality of the segmentation depends on the quality of the extracted centerline, and the gradients produced by the and gradient vector flow. If the GVF contains

a high degree of noise, or ”variations” within a small area, the segmentation result will contain edges and other abnormalities that are not vessel-like. This can be partially solved by applying smoothing to the GVF, or by running a high number of iterations in the GVF.

If a centerline produced either by the ridge traversal, or the grouping and linking, or both, for some reason is outside a vessel, or in a position where no vessel is detected, the segmentation can contain holes/gaps. This can also happen if the centerline is to far off-center. The off-center problem can be partially solved by applying more initial dilation. But that might result in over segmenting some of the smaller vessels. The holes can also be fixed by applying some sort of post-processing patching.

5.6 Run-times

The most significant factor affecting the total run-time is the gradient vector flow. With 500 iterations they require a significant amount of time ($\approx$ 9min) to compute. While the total run time is around 11min depending on the size of the initial CTA image. The second most significant part of our implementation in terms of run-time performance is the ridge traversal, candidate point detection and grouping and linking. The ridge traversal is very much dependent on the amount of candidate points it has to process, while the candidate detection is dependent on the amount of points it needs to detect. In our implementation most CTA images both from the testing framework, and from the St.Olavs datasets where run with $\approx$ 500 candidate points. The linking and grouping process can be time consuming if there are a lot of vessels with potential connections.

Our testing PC has an NVIDIA GPU. NVIDIA, as stated earlier, does not support writing to 3D textures. This probably has a great impact on the run-time, as writing to 3D textures is considered to be significantly faster. AMD GPUs does support writing to 3D structures, but unfortunately we did not have an AMD GPU to test this on.

Chapter 6

Conclusion

6.1 Goal Achievement

The purpose of this project was to implement a program for coronary artery centerline extraction and segmentation that utilizes the computational power of GPUs. To select an appropriate method we conducted a wide background study to identify methods that fit the requirements for this project. We chose one of the most promising methods, and implemented it in *FAST*, using OpenCL and C++. We have tested our implementation and shown that it able to extract large parts of the coronary arteries.

6.2 Future Work

In the previous chapter we discussed several weaknesses and problems with our implementation. The potential for incorrect linking and grouping, the lack of robustness for the ridge traversal, and the potential for partial, or broken centerline production because of noise or low TDF responses are some of the main issues.

6.2.1 Pre-processing

As mentioned earlier, NVIDIA GPUs does not support writing to 3D textures through OpenCL. This has a major impact on the time required to process some parts of our implementation. The run-time of GVF would benefit significantly if the GPU utilized for the processing is an AMD GPU. The gradient vector field, both TDFs, and both Gaussian smoothing and Non-local Means would also benefit from using a AMD GPU, but not as much as the GVF would.

More noise reduction could also benefit the results produced. Less noise would result in a more stable and reliable result, even if that could mean that the run-time is increased. In some cases it can be difficult to determine the quality of both the centerline and the segmentation produced without manually looking at the CTA and the centerlines and segmentation. So if a more intelligent approach to noise reduction could result in a more reliable centerline/segmentation it could be worth it. There are several methods that could provide this, a anisotropic filter presented by Bauer[5], a Bayesian filter by Snaches et al[40], a hybrid diffusion filter by Mendrik A.M et al[32], and many more could be suited for this task.

6.2.2 TDFs and Grouping and Linking

Both the TDFs used in our implementation has the weakness that they return low, or negative response in junctions or in vessels with a very irregular shape. This definitely is a much more significant problem with the Circle Fitting TDF, but the Vesselness Filter does have a slight problem with this as well. The grouping and linking approach implemented here reduces the significance of this problem, but as discussed in chapter 5.4.2 it could lead to incorrect connections between vessels, or inaccurate centerlines. A possible solution to this could be to utilize a third TDF for detection of junctions and irregular vessels, and use this TDF in addition to the Vesselness filter in the ridge traversal. Then the grouping and linking approach could be utilized to connect some of the smaller vessels if needed.

6.2.3 Ridge Traversal

The ridge traversal relies on the TDF responses. If the TDF response is low, or wrong, the ridge traversal might just stop or produce an inaccurate centerline. Detecting when the ridge traversal does this without providing manual aid is a difficult task. The approach to deal with this by Bauer et al[6, 5] is to simply use many candidate points, and in that way produce as many centerlines as possible, and then use linking and grouping to connect them if needed. This can solve the problem where the ridge traversal terminates to early, but it does not solve the potential for inaccurate centerlines.

A possible solution could be to include one of the methods discussed in the background study, or to simply replace the ridge traversal approach. Many of the methods discussed in the background study utilizes candidate points in their extraction of centerlines, and the TDFs could be used to generate them. The region growing

approach presented by Krissian et al[28], or the the refined Vesselness approach by Yang et al[53] could be good alternative methods, or complimentary methods to create a more reliable centerline extraction.

6.2.4 Centerline Selection

The centerline selection method used in our implementation is fairly primitive with a significant weakness. The weakness (as mentioned in chapter 5.4.1) is that is selects the longest, or the top 3-4 longest centerlines produced. If the ridge traversal and grouping and linking produces several disconnected centerlines, the result of the selection might return vessels other than the coronary arteries. Zheng et al[54] utilizes an limitation approach where they locate the heart in the CTA image, and select the centerlines that are close to the heart location. This approach seems to be far more robust then relying on the length of the centerlines.

6.2.5 Segmentation

The inverse gradient tracking segmentation is reliant on the quality and correctness of the centerlines produced. It is also susceptible to noise, if the noise affects the gradients produced by GVF. But if the centerline is good, the method works fairly well. An alternative solution could be a region growing approach (Krissian et al[28]), but if this approach produces better segmentations then the approach presented here, is uncertain. A more parallel approach would also benefit the overall run-time, as the inverse gradient tracking algorithm is executed serially. But the time required to perform the segmentation compared to the time GVF and the centerline extraction uses is very low.

Bibliography

[1] Dierker J. Joite-Barfuss S. Aichinger, H. and M Sabel. *Radiation Exposure and Image Quality in X-Ray Diagnostic Radiology*. Dordrecht: Springer, 2011.

[2] Pizer S. Bullitt-E. Aylward, S. and D Eberl. Intensity ridge and widths for tubular object segmentation and description. *Proceedings of the Workshop on Math. Methods in Biomed. Image Analysis*, 1(1):131–138, 1996.

[3] S. Aylward and E Bullitt. Analysis of the parameter space of a metric for registering 3d vascular images. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2001.

[4] G. W. Barsness and D. R. Holmes. *Coronary Artery Disease*. Springer, 2011.

[5] C. Bauer. *Segmentation of 3D Tubular Tree Structures in Medical Images*. PhD thesis, Graz University of Technology, 2010.

[6] C. Bauer and H. Bischof. Edge based tube detection for coronary artery centerline extraction. *Midas Journal*, July 2008.

[7] T. van Walsum A. van der Giessen A. Weustink N. Mollet G. Krestin C. Metz, M. Schaap and W. Niessen. 3d segmentation in the clinic:a grand challenge ii - coronary artery tracking. In *MICCAI*, MICCAI'08, 2008.

[8] J Canny. inding edges and lines in images. *Tech. Rep. 720, MITAIL*, 1983.

[9] S. Cetin and G Unal. A higher-order tensor vessel tractography for segmentation of vascular structures. *Medical Imaging, IEEE Transactions*, 34(10):2172–2185, 2015.

[10] R Cierniak. *X-Ray Computed Tomography in Biomedical Engineering*. Dordrecht: Springer, 2011.

[11] Wikimedia Commons. Echocardiography, sonography, ultrasonography, medical ultrasound, cardiology, vsd, ventricular septal defect, 2005.

[12] Wikimedia Commons. bolus tracking in the use of imaging an abdominal aortic aneurysm, 2007.

[13] Wikimedia Commons. Coronary arteries and coronary circulation, 2010.

[14] Wikimedia Commons. Placement of the precordial leads in electrocardiography, 2012.

[15] Doosthoseini A Dehkordi M, Sadri S. A review of coronary artery segmentation algorithms. *Jurnal of Medical Signals and Sensors*, 1(1):49–54, 2011.

[16] F Filipoiu. *Atlas of Heart Anatomy and Development*. Dordrecht: Springer, 2013.

[17] Rotterdam Coronary Artery Algorithm Evaluation Framework. Main page, June 2016.

[18] Rotterdam Coronary Artery Algorithm Evaluation Framework. Results, June 2016.

[19] Alejandro F. Frangi, Wiro J. Niessen, Koen L. Vincken, and Max A. Viergever. *Multiscale vessel enhancement filtering*. Springer Berlin Heidelberg, 1998.

[20] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

[21] O. Friman, C. Kuehnel, and H. Peitgen. Coronary centerline extraction using multiple hypothesis tracking and minimal paths. *Midas Journal*, July 2008.

[22] Howes L. Kaeli D. Mistry P. Gaster, B. and D Schaa. *Heterogeneous Computing with OpenCL : Revised OpenCL 1.2 Edition (2nd ed.)*. Burlington: Elsevier Science, 2012.

[23] Anand S Abrahams-Gessel S Murphy A Gaziano TA, Bitton A. Growing epidemic of coronary heart disease in low- and middle-income countries. *Current problems in cardiology*, 35(2):72–115, 2010.

[24] Khronos Group. Main page, June 2016.

[25] Quek F Kirbas C. A review of vessel extraction techniques and algorithms. *AMC Computing Surveys*, 36(2):81–121, 2004.

[26] Malandain G. Krissian, K. and N Ayache. Directional anisotropic diffusion applied to segmentation of vessels in 3d images. *Tech. Rep. 3064, INRIA*, 1996.

[27] Malandain G. Ayache N Krissian, K. Model-based detection of tubular structures in 3d images. *Computer Vision and Image Understanding*, 80(2):130–171, 2000.

[28] Pozo J.M. Villa-Uriol M.C. Frangi A Krissian K., Bogunovic H. Minimally interactive knowledge-based coronary tracking in cta using a minimal cost path. *Midas Journal*, July 2008.

[29] David Lesage, Elsa D. Angelini, Isabelle Bloch, and Gareth Funka-Lea. *Bayesian Maximal Paths for Coronary Artery Segmentation from 3D CT Angiograms*. Springer Berlin Heidelberg, 2009.

[30] Reinders J. McCool, M. and A Robison. *Structured Parallel Programming : Patterns for Efficient Computation.* Burlington: Elsevier Science, 2012.

[31] Peter A McCullough. Coronary artery disease. *Clinical Journal of the American Society of Nephrology*, 2(3):611–616, 2007.

[32] Rutten A Viergever MA van Ginneken B Mendrik AM, Vonken EJ. Noise reduction in computed tomography scans using 3-d anisotropic hybrid diffusion with continuous switch. *IEEE Transactions on Medical Imaging*, 28(10):1585–1594, 2009.

[33] Kawatak Y. Sato H. Niki, N. and T Kumazaki. 3d imaging of blood vessels using x-ray rotational angiographic system. *IEEE Med. Imaging Conf*, 3(1):1873–1877, 1993.

[34] World Health Organization. Cardiovascular diseases (cvds), June 2016.

[35] J. F. O'Brien and N. F Ezquerra. Automated segmentation of coronary vessels in angiographic image sequences utilizing temporal, spatial structural constraints. In *Proc. SPIE Conf. Visualization in Biomed. Computing*, 1994.

[36] Elion J. Petrocelli, R. and K. M Manbeck. A new method for structure recognition in unsubtracted digital angiograms. *IEEE Computers in Cardiology*, page 207–210, 1992.

[37] Morse B. Pizer, S. and D Fritsch. Zoominvariant vision of figural shape: the mathematics of cores. *Computer Vision and Image Understanding*, 69(1):55–71, 1998.

[38] R. Poli and G Valli. An algorithm for realtime vessel enhancement and detection. *Comp. Methods and Prog. in Biomed*, 52(1):1–22, 1997.

[39] Monga O. GE C. Xie S. Prinet, V. and S Ma. Thin network extraction in 3d images: Application to medical angiograms. *Proc. Int. Conf. Pattern Rec*, page 386–390, 1996.

[40] Joao Miguel Sanches, Luisa Micó, and Jaime Cardoso. *Pattern Recognition and Image Analysis*. Springer Berlin Heidelberg, 2013.

[41] Neefjes Metz Capuano De Bruijne Schaap, Van Walsum and Niessen. Robust shape regression for supervised vessel segmentation and its application to coronary segmentation in cta. *Medical Imaging, IEEE Transactions*, 30(11):1974–1986, 2011.

[42] Verbeeck G. Suetens P. Smets, C. and A Oosterlinck. A knowledge-based system for the delineation of blood vessels on subtraction angiograms. *Pattern Rec. Lett*, 8(1):113–121, 1988.

[43] Bozorgi M. Lindseth F. Smistad, E. Fast: Framework for heterogeneous medical image computing and visualization. *International Journal of Computer Assisted Radiology and Surgery*, 10(11):1811–1822, 2015.

[44] Halmai C. Erdohelyi B. Palagyi K. Nyul L. Olle K. Geiger B. Lindbichler F. Friedrich G. Sorantin, E. and K Kiesler. Spiral-ctbased assessment of tracheal stenoses using 3-dskeletonization. *IEEE Trans. on Med. Img*, 21(3):263–273, 2002.

[45] P. Summers and A Bhalerao. Derivation of pressure gradients from magnetic resonance angiography using multi-resolution segmentation. *International Conference on Image Processing and its Applications*, page 404–408, 1995.

[46] A. Szymczak. Vessel tracking by connecting the dots. *The MIDAS Journal*, 2008.

[47] J Thie. *Nuclear Medicine Imaging : An Encyclopedic Dictionary*. Dordrecht: Springer, 2012.

[48] P. Viola and M Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, Proceedings of the 2001 IEEE Computer Society Conference*, 2001.

[49] Wilson R. Vlodaver, Z. and D Garry. *Coronary Heart Disease : Clinical, Pathological, Imaging, and Molecular Profiles*. Springer, 2012.

[50] Harvey E. Cline William E. Lorensen. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), 1987.

[51] Nathan D Wong. Epidemiological studies of chd and the evolution of preventive cardiology. *Nature Reviews Cardiology*, 11(5):276–289, 2014.

[52] Prince J Xu, C. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998.

[53] Kitslaar P. Frenay M. Broersen A. Boogers M. Bax J. . . . Dijkstra J Yang, G. Automatic centerline extraction of coronary arteries in coronary computed tomographic angiography. *The International Journal of Cardiovascular Imaging*, 28(4):921–933, 2011.

[54] Huseyin Tek Yefeng Zheng and Gareth Funka-Lea. Robust and accurate coronary artery centerline extraction in cta by combining model-driven and data-driven approaches.

[55] Yuanzhong Li Yoshiro Kitamura and Wataru Ito. Automatic coronary extraction by supervised detection and shape matching.