Noname manuscript No.
(will be inserted by the editor)

GPU Accelerated Segmentation and Centerline Extraction of Tubular Structures from Medical Images

Erik Smistad · Anne C. Elster · Frank Lindseth

the date of receipt and acceptance should be inserted later

Abstract
Purpose To create a fast and generic method with sufficient quality for extracting tubular structures such as blood vessels and airways from different modalities (CT, MR and US) and organs (brain, lungs and liver) by utilizing the computational power of graphic processing units (GPUs).
Methods A cropping algorithm is used to remove unnecessary data from the datasets on the GPU. A model-based tube detection filter combined with a new parallel centerline extraction algorithm and a parallelized region growing segmentation algorithm is used to extract the tubular structures completely on the GPU. Accuracy of the proposed GPU method and centerline algorithm is compared to the ridge traversal and skeletonization/thinning methods using synthetic vascular datasets.
Results The implementation is tested on several datasets from three different modalities: airways from CT, blood vessels from MR and 3D Doppler Ultrasound. The results show that the method is able to extract airways and vessels in 3-5 seconds on a modern GPU and is less sensitive to noise than other centerline extraction methods.
Conclusions Tubular structures such as blood vessels and airways can be extracted from various organs imaged by different modalities in a matter of seconds, even for large datasets.

Keywords Segmentation · Centerline extraction · Vessel · Airway · GPU · Parallel

Erik Smistad · Anne C. Elster · Frank Lindseth
Dept. of Computer and Information Science
Norwegian University of Science and Technology
Sem Saelandsvei 7-9, NO-7491 Trondheim
Tlf.: +47 73594475
E-mail: smistad@idi.ntnu.no

Frank Lindseth
SINTEF Medical Technology

1 Introduction

Blood vessels and airways are both examples of important tubular structures in the human body. The extraction of these structures can be essential for planning and guidance of several surgical procedures such as bronchoscopy, laparoscopy and neurosurgery.

Registration is to create a mapping between two domains, for instance between an image and the patient or between different image modalities [34]. Registration is an important step in image guided surgery as it enables us to accurately plot the location of surgical tools inside the body onto images of the patient using optical or magnetic tracking technology. Tubular structures extracted from preoperative images can be matched to similar intraoperative structures, e.g. airways generated by a tracked bronchoscope or brain vessels extracted from power Doppler based 3D ultrasound, and consequently create the mapping between preoperative images and the patient. Also, extracted tubular structures from pre- and intraoperative image data can be used to reduce registration errors when a corresponding point (anatomical landmarks or fiducials) patient registration method is used.

Furthermore, during surgical procedures, anatomical structures have a tendency to move and deform inside the body due to respiration, pulsation, external pressure and resection. This is called anatomical shift and is a major challenge as it reduces the surgical navigation accuracy. However, it has been shown that registration of blood vessels from pre- and intraoperative image data can be used to detect and correct organshift such as brainshift [38].

The automatic extraction of tubular structures can be very time consuming. As time during surgery is very crucial, long-lasting processing should be avoided. Preoperative data

is often acquired just before the procedure and thus it is desirable to process these data as fast as possible as well. The purpose of this work is to create a fast and generic method with sufficient quality for extracting tubular structures such as blood vessels and airways from different modalities (CT, MR and US) and organs (brain, lungs, liver) by utilizing the computational power of graphic processing units (GPUs).

The rest of the introduction discuss GPU computing and provides a brief survey of existing methods for extracting tubular structures from medical images. An overview of the contributions in this paper is also given. The methodology section provides a detailed description of each part of the implementation including how it is optimized for the GPU and evaluated. In the result section, performance is measured in terms of speed and quality. Finally, the results are discussed and conclusions are given.

1.1 GPU computing

Several image processing techniques are data parallel because each pixel can be processed in parallel using the same instructions. Graphic Processing Units (GPUs) allow many pixels/voxels to be processed in the same clock cycle, enabling substantial speedups. The GPU is a type of single instruction, multiple data (SIMD) processor. It can perform the same instruction on each element in a dataset in parallel. This is achieved by having many functional units like arithmetic-logic units (ALUs) that share a control unit. Fig. 1 depicts the general layout of a GPU and its memory hierarchy. The GPU originally had a fixed pipeline that was created for fast rendering of 3D graphics. The introduction of programmable shaders in the pipeline made it possible to run programs on the GPU. However, the task of programming shaders to solve arbitrary problems requires knowledge about the GPU pipeline as the problem at hand needs to be transformed into a rendering problem. General purpose GPU (GPGPU) programming languages and frameworks such as CUDA and OpenCL were created to make GPU programming easier. The field of GPU computing is still young. However, a brief survey of medical image processing and visualization on the GPU was recently provided by Shi et al. [39].

1.2 Methods for extracting tubular structures

Tubular structures are usually extracted from volumes in two different ways:

- As a **segmentation**, either as a binary classification where each voxel in the volume is given a non-zero value if it belongs to the tubular structure or as a surface model of the structure.

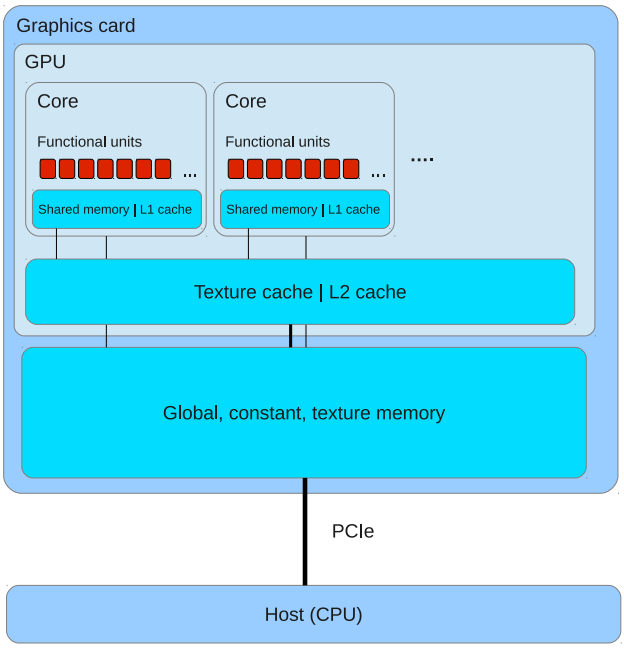

Fig. 1 General architecture of a GPU and its memory hierarchy. Note however, that the actual architecture is much more complex and differ for each GPU. This diagram only shows the general features.

- As a **centerline**, i.e. a line that goes through the center of the tubular structures.

Both representations are useful in different applications. For instance, the centerline is very useful for registration while the segmentation is useful for volume estimation and visualization of the structures' surface.

There exist several methods for extracting tubular structures from medical images. A recent and extensive review on blood vessel extraction was done by Lesage et al. [29] and an older one by Kirbas and Quek [25]. Two reviews on the segmentation of airways were done by Lo et al. [31] and Sluimer et al. [40].

A common method for extracting tubular structures is to grow the segmentation iteratively from an initial point or area using methods such as region growing [27,45,16], active contours and wave front propagation (e.g. snakes and level sets) [24,35,46,32]. A centerline can then be extracted from the segmentation using skeletonization and 3D thinning methods [28,18,22].

Growing a segmentation using only a model of desired intensity values has shown to give limited result in several applications such as airway segmentation where the thin airway walls may cause severe segmentation leakage [32]. Thus in many applications it may be necessary to use a model of the shape of the tubular structures as well. Also, these growing methods are very sensitive to initialization.

Tube Detection Filters (TDFs) are used to detect tubular structures and calculates a probability that a specific voxel

is inside a tubular structure. Most TDFs use gradient information, often in the form of an eigenanalysis of the Hessian matrix. Frangi et al. [15] presented an enhancement and detection method for tubular structures based on the eigenvalues of this matrix. Krissian et al. [26] created a model-based detection filter that fits a circle to the cross-sectional plane of the tubular structure defined by the eigenvectors of the Hessian.

A centerline can be extracted directly from the TDF result without a segmentation using methods such as ridge traversal. Aylward et al. [2] provides a review of different centerline extraction methods and proposed an improved ridge traversal algorithm based on a set of ridge criteria and different methods for handling noise. Bauer et al. [6] showed how this method could be used together with Gradient Vector Flow. For applications where only the centerline is needed, segmentation can be skipped using this method and thus reduce processing time.

Some related work on accelerating the extraction of tubular structures on the GPU exist. Erdt et al. [14] performed the TDF and a region growing segmentation on the GPU and reported a 15 times faster computation of the gradients and up to 100 times faster TDF. Narayanaswamy et al. [36] did vessel luminae region growing segmentation on the GPU and reported a speedup of 8. Bauer et al. presented a GPU acceleration for airway segmentation by doing the Gradient Vector Flow computation on the GPU in [7] and the TDF calculation on the GPU in [8]. However, they only provide a limited description of the GPU implementations. Helmberger et al. performed region growing for airway segmentation on the GPU and a lung vessel segmentation on the GPU using a TDF [21]. They reported a runtime of 5-10 minutes using a modern GPU and CUDA compared to a runtime of up to an hour using only the CPU.

1.3 Contributions

The methodology in this paper is inspired by the works of Bauer et al. [7,3,8] and Krissian et al. [26] and is a continuation of our previous paper on GPU accelerated airway segmentation [41].

The main contributions in this paper are:

- A fast and generic method that can extract tubular structures like blood vessels and airways from different modalities (e.g. CT, MR and Ultrasound) and organs (e.g. lung, brain and liver) entirely on the GPU.
- A new parallel GPU algorithm for extracting centerlines directly from the TDF result.
- A generic parallel cropping algorithm for reducing memory usage on the GPU.

2 Methodology

The implementation is written in C++ and OpenCL and is available online. OpenCL is a framework for running parallel programs on heterogeneous platforms such as CPU and GPU. The implementation consists of five main steps that are all executed on the GPU (see Fig. 2).

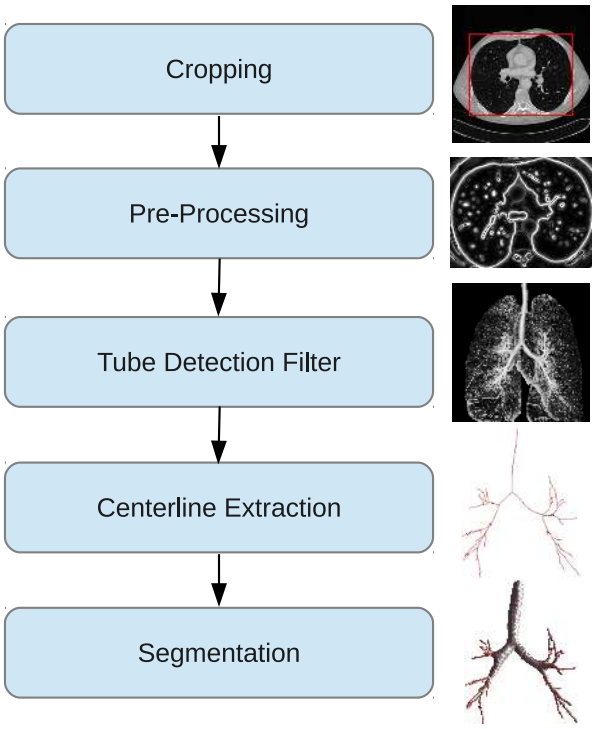

Fig. 2 Block diagram of the implementation

The first step is to crop the volume in order to reduce the total memory usage. The second step involves a few preprocessing steps such as Gaussian smoothing and Gradient Vector Flow which are necessary to make the results less sensitive to noise and differences in tube contrast and size. After pre-processing, the model-based TDF by Krissian et al. [26] is used. From the TDF result, the centerlines are extracted using a new parallel algorithm. Finally, a segmentation is performed using the centerlines as seeds for a region growing procedure. However, if only the centerlines are needed for a given application, the segmentation step can be skipped. The rest of this section will describe each of the five steps in further detail.

2.1 Cropping

Memory on the GPU is limited and may not be enough for processing large datasets. However, most medical datasets

http://github.com/smistad/Tube-Segmentation-Framework/

contain a lot of data that is not part of the structures of interest. Usually these areas are located at the borders of the image. For instance, in the thorax CT image in Fig. 3, the actual lungs where the airways and blood vessels are located, constitutes only about 50% of the image. The rest consist of space outside the body, body fat and the bench that the patient is resting on. As several of the methods used to perform segmentation and centerline extraction process each voxel in the entire volume, removing the unnecessary data will not only reduce memory usage, but also execution time.

In our previous work [41], we presented a novel cropping algorithm for airway segmentation that could be run in parallel on the GPU using less than half a second for large CT volumes of the lungs. In this paper, this algorithm is extended to crop other medical datasets, such as MR and 3D Doppler Ultrasound. The cropping method works by considering slices in all three orthogonal directions x, y and z. For each slice s, the method determines if the slice intersects the region of interest (ROI). This is done by counting the number of rows in the slice that intersects the ROI for each slice and storing it as L_s. If $L_s > L_{\text{min}}$, the slice is considered to have intersected the ROI. The cropping borders are found by traversing through L_s twice from $s = 0$ and $s =$ size and finding the first slice that has a value above a specific threshold L_{min}. These slices are then selected as cropping borders c_1 and c_2. This is done for each direction and results in 3 pairs of cropping borders which is all that is needed to crop the volume. An example of how this cropping procedure works is shown in Fig. 3. For some applications and directions it may be necessary to start the search from the middle $s = \frac{\text{size}}{2}$ to the end instead. This was the case for the axial direction of CT airway datasets.

Algorithm 1 provides pseudocode for the cropping method. The function CALCULATEL is used for estimating L for each slice in a given direction and the function FINDCROPBORDERS is used to find the cropping borders for a specific direction given L and using the threshold L_{min}. Each direction and slice can be processed in parallel on the GPU. For a dataset of size 512x512x512 this results in 3*512 individual threads that can be processed using the same instructions.

The parts of this cropping method that is application dependent, aside from the parameter L_{min}, is the estimation of L_s and whether the search for cropping borders starts from the middle or at the ends of the dataset in a given direction.

For MR and 3D Doppler Ultrasound images it is sufficient to remove the background from the dataset. For this purpose L_s can be estimated by counting the number of voxels n_v on a scan line that is above a certain threshold I_T. L_s is then set to the number of rows in slice s where $n_v > I_T$. A threshold value of 100 was found to be enough for the MR and Ultrasound datasets.

For CT images of the lungs, fat and other tissue that are not part of the lungs can also be discarded by counting the number of areas that are above and below a certain threshold. Details on this estimation of L_s can be found in our previous work [41].

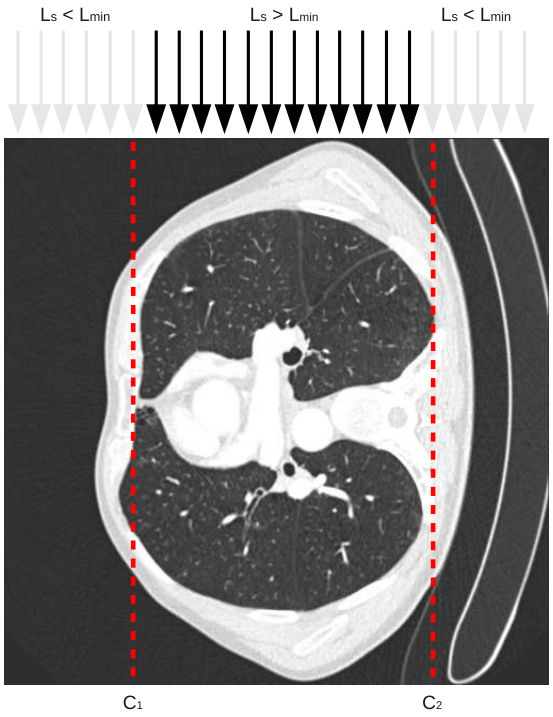

Fig. 3 Example of the cropping procedure. The black arrows indicate slices that have $L_s > L_{\text{min}}$ and thus intersected the ROI while the grey arrows are the opposite. This can be used to find the cropping borders, marked with dotted red lines in the figure. This is done in all three directions and each slice is processed in parallel.

2.2 Pre-processing and Gradient Vector Flow

Before the actual tube extraction, some pre-processing is necessary. First, an optional thresholding is performed on the dataset using a lower and upper threshold (I_{min} and I_{max}). Thresholding may be necessary for datasets which have a large range of intensity values such as CT images. The thresholding is done to remove unnecessary gradient information in the image which may lead to unwanted tubular structures being detected. For instance, when extracting airways all intensities above -500 HU can be converted to -500 as no airways have intensity above this threshold. Second, some noise suppression is performed. This is done by blurring the dataset using Gaussian smoothing with standard deviation σ. Afterwards, the gradient vector field $\mathbf{V}$ is created and normalized using a parameter called V_{max}. All gradients with a length above this parameter will be set to unit length and the others will be scaled accordingly. The gradient normalization is necessary for contrast invariance. V_{max} should be adapted to the expected level of contrast and noise. Also, if black tubular structures are to be extracted (e.g. airways),

Algorithm 1 Cropping

```
function CROP(volume)
    L ← CALCULATEL(volume, x)
    x1, x2 ← FINDCROPBORDERS(L, x)
    L ← CALCULATEL(volume, y)
    y1, y2 ← FINDCROPBORDERS(L, y)
    L ← CALCULATEL(volume, z)
    z1, z2 ← FINDCROPBORDERS(L, z)
    crop volume according to x1,x2,y1,y2,z1 and z2
     return volume
end function

function CALCULATEL(volume, direction)
    for each slice s in direction in parallel do
        Estimate Ls
    end for
     return L
end function

function FINDCROPBORDERS(L, direction)
    size ← volume.direction.size
    c1 ← −1, c2 ← −1, s ← 0
    while (c1 = −1 or c2 = −1) and s < size do
        if Ls > Lmin and c1 = −1 then
            c1 ← s
        end if
        if Lsize−1−s > Lmin and c2 = −1 then
            c2 ← size −1− s
        end if
        s ← s+1
    end while
     return c1, c2
end function
```

the gradients have to inverted $\mathbf{V} = -\nabla I$. All of these preprocessing parameters ($I_{\min}$, $I_{\max}$, σ, $V_{\max}$) are modality dependent and the values used in this paper for each modality is collected in Table 1.

Filters that use the Hessian matrix to detect tubular structures require gradient information to be present in the center of the tube. For large tubes, such as *trachea* and the main *bronchi*, the gradient information will not exist in the center. Thus, it is necessary to propagate the gradient information from the tube edge to the center. There exist two main methods of doing this: Gaussian scale space and Gradient Vector Flow (GVF). Xu et al. [47] originally introduced GVF as an external force field for active contours. Bauer et al. [5, 3] were the first to show that GVF could be used to create scale-invariance of TDFs. The GVF method has the advantage that it is feature-preserving and thus can avoid the problem of several structures diffusing into each other to create the illusion of a tubular structure at a higher scale. Also, GVF is only calculated using one scale. However, it has the disadvantage that it is very computationally expensive. Nevertheless, it has been shown that GVF can be accelerated using GPUs. Eidheim et al. [13], He and Kuester [20] and Zheng and Zhang [48] all presented a GPU implementation of GVF and Active Contours using shader languages. However, their implementation was for 2D images only. In this paper, a highly optimized 3D GPU implementation of GVF from Smistad et al. [42] was used with a predefined number of 250 iterations. This implementation allows GVF to be calculated for large volumes in only a few seconds.

2.3 Tube Detection Filter

Krissian et al. [26] created a TDF that assumes that the cross-section of the tubular structure is circular. Their TDF calculates how well a circle match the gradient information in the cross-sectional plane defined by the eigenvectors of the Hessian matrix. The TDF starts by creating a circle with a small radius in the cross-sectional plane. $N = 32$ evenly spaced points on the circle is sampled from the vector field. Each point, i, is found by calculating its angle α from the center and then calculating a vector $\mathbf{d}_i$ which lies in the plane and has angle α.

$$\alpha = \frac{2\pi i}{N} \tag{1}$$

$$\mathbf{d}_i = \mathbf{e}_2 \sin\alpha + \mathbf{e}_3 \cos\alpha \tag{2}$$

The position of point i on a circle with radius r and center $\mathbf{v}$ is then given as $\mathbf{v} + r\mathbf{d}_i$. How well the circle match the gradient information is calculated as the average dot product of the gradient at position i and the inward normal of the circle at point i which is equal to $-\mathbf{d}_i$. The TDF of Krissian et al. [26] is shown in equation 3. The radius of the circle is increased with 0.5 voxels as long as the average dot product also increases.

$$T(\mathbf{v}, r, N) = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{V}(\mathbf{v} + r\mathbf{d}_i) \cdot -\mathbf{d}_i \tag{3}$$

As noted by Bauer et al. [7,3], the GVF method may eliminate the gradient information for small low-contrast tubular structures. Thus to detect these tubular structures it is necessary to run the TDF two times. Once with a small radius on the initial vector field to detect the small low-contrast structures and once with the GVF vector field to detect the rest. Different amounts of Gaussian blur can be used for the tube detection of large and small structures (σ_{small} and σ_{large} as seen in Table 1). The TDF response from each of these are combined by selecting the largest TDF value for each voxel.

2.4 Centerline Extraction

Centerline extraction from TDF results has primarily been done by ridge traversal [2,4,6,5]. One problem with the ridge traversal procedure is that it can't be run in parallel. Thus,

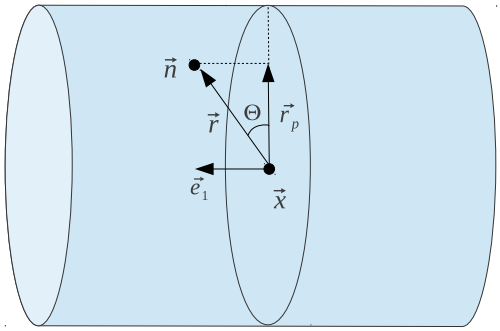

Fig. 4 Determining the angle θ from a centerpoint $\mathbf{x}$ to its neighbor $\mathbf{n}$.

the GVF vector field and the TDF result has to be transferred to the CPU. Nevertheless, the serial ridge traversal algorithm can be used together with the rest of the GPU algorithms presented in this paper (e.g. cropping, pre-processing, tube detection and segmentation).

In this section, a new parallel centerline extraction (PCE) algorithm is presented. This centerline algorithm, unlike ridge traversal, can be run efficiently in parallel on a GPU. The method has 4 main steps: Identifying centerpoints, filtering centerpoints, link centerpoints and centerline selection.

2.4.1 Identify candidate centerpoints

The method for extracting centerlines starts by identifying all possible centerpoints. This is done by creating a 3D structure with the same size as the dataset. This structure is initialized to 0 for each voxel and all voxels with a TDF value above the threshold $T_c = 0.5$ is set to 1.

2.4.2 Filter centerpoints

The next step removes centerpoints that are either not in the center of a tube or too close to other centerpoints. Whether a centerpoint is in the center of a tube or not can be determined by the magnitude of the GVF vector field $|\mathbf{V}|$, because $|\mathbf{V}|$ is smallest in the center of the tube.

First, a vector from the centerpoint $\mathbf{x}$ to a neighbor voxel $\mathbf{n}$ is calculated:

$$\mathbf{r} = \mathbf{n} - \mathbf{x} \tag{4}$$

Second, this vector is projected onto the cross-sectional plane of the tube (see Fig. 4). The plane's normal $\mathbf{e}_1$ is the eigenvector of the Hessian matrix associated with the eigenvalue of smallest magnitude. This vector points in the direction of the tube.

$$\mathbf{r}_p = \mathbf{r} - \mathbf{e}_1(\mathbf{e}_1 \cdot \mathbf{r}) \tag{5}$$

Finally, the angle θ from the plane to the vector $\mathbf{r}$ can be calculated using the projected vector $\mathbf{r}_p$:

$$\theta = \cos^{-1}\left(\frac{\mathbf{r} \cdot \mathbf{r}_p}{|\mathbf{r}||\mathbf{r}_p|}\right) \tag{6}$$

Let $\mathbf{N}$ be the set of all neighbor voxels that are close ($|\mathbf{r}| < r$, where r is from Eq. 3) and the angle is $\theta < 30°$. For each of these $\mathbf{n}$, the magnitude of the GVF vector field $|\mathbf{V}|$ is compared to the centerpoint $\mathbf{x}$. The centerpoint is only valid if the magnitude for the centerpoint $\mathbf{x}$ is lower than all $\mathbf{n} \in \mathbf{N}$:

$$C(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \forall \mathbf{n} \in \mathbf{N} \quad |\mathbf{V}(\mathbf{n})| > |\mathbf{V}(\mathbf{x})| \\ 0 & \text{else} \end{cases} \tag{7}$$

This has the effect that it removes centerpoints that are not in the center of a tubular structure.

The next step is to remove centerpoints that are too close to each other. The reason for doing this is that it reduces the total number of centerpoints and thus makes the next step, linking the centerpoints, much more efficient. Removing points that are too close to each other is done by dividing the entire dataset into a grid with each grid element spanning 4x4x4 voxels. For each cube in the grid, the best centerpoint is selected and the rest of the centerpoints in that cube is removed. The centerpoint with the highest TDF value is selected as the best centerpoint in a cube.

2.4.3 Link centerpoints

For each centerpoint, the method establishes links between the centerpoints to create centerlines. This is done by connecting each centerpoint to the two centerpoints that are closest and fulfills the following criteria:

- The angle between them is above 120 degrees.
- The average TDF value along the line is higher than $T_{\text{mean}} = 0.5$.

2.4.4 Centerline selection

Due to noise and other image artifacts invalid centerpoints and centerlines may be created. However, these are usually short and not connected to the actual tubular structures. Thus invalid centerlines can often be discarded based on their length.

In this step, all centerpoints that are connected with centerlines from the previous section are assigned the same label. Those that are not connected get different labels. Graph component labeling is the problem of finding and labeling nodes in a graph that are connected. Hawick et al. [19] presented several GPU implementations of algorithms for graph component labeling. In our implementation, an iterative method using atomic operations was used. Assuming N labels, N

counters are created and initialized to 0. A kernel is executed for each centerpoint and the length of each centerline, identified with a label, is determined by using an atomic increment operation on the counter identified by the centerpoints' labels. After the execution of this kernel, the counters will contain the total length of each centerline. When the length of all connected centerlines have been calculated, the largest centerline or all centerlines with a specified minimum length can be extracted.

2.5 Segmentation

Bauer et al. [7] proposed a method for generating a segmentation from the centerline using the already computed GVF vector field. They named this method Inverse Gradient Flow Tracking Segmentation because it for each voxel tracks the centerline using the directions of the GVF vector field, but in the inverse direction. This segmentation method is a type of seeded region growing, where the centerlines are the seeds and the direction and magnitude of the vectors from the GVF vector field is used to determine if the segmentation is allowed to continue to grow.

In this paper, a data parallel version of this algorithm is presented (see Algorithm 2). First, the centerlines, C, are dilated in parallel on the GPU and added to the segmentation *S*. Next, the neighboring voxels of *S* is added to a queue *Q*. For each iteration, the GROW function runs a kernel on each voxel $\mathbf{x}$ in the entire volume. If the voxel $\mathbf{x}$ is part of *Q*, the gradients of all unsegmented neighbors are checked to see if they point to $\mathbf{x}$ and has a larger magnitude than $\mathbf{x}$. If such a neighbor voxel $\mathbf{y}$ is found, $\mathbf{x}$ is added to *S*, its neighbor $\mathbf{y}$ is added to *Q* and the stopGrowing variable is set to false. Since this variable is initialized to true for every iteration, the growing procedure will stop when no more voxels are added.

For the 3D Ultrasound Doppler modality another segmentation method than the inverse gradient tracking method is used. The reason for this, is that this data can be quite noisy. This alternative segmentation method starts by calculating an average radius based on the circle fitting method for each link. For each discrete point on the centerline, all voxels within a sphere with the same radius is marked as part of the segmentation.

2.6 GPU Optimization

2.6.1 Texture system

The GPU has a specialized memory system for images, called the texture system. The GPU has this because the GPU is primarily made and used for fast rendering which involves mapping images, often called textures, onto 3D objects.

Algorithm 2 Parallel Inverse Gradient Flow Tracking

```
S ← DILATE(C)
Q ← DILATE(S) - S
stopGrowing ← false
while !stopGrowing do
    stopGrowing ← true
    GROW(S, Q, stopGrowing)
end while
return S

function GROW(S, Q, stopGrowing)
    for each voxel x in parallel do
        if x ∈ Q then
            for each voxel y ∈ Adj26(x) do
                if y ∉ S and |V(y)| > |V(x)| and
                  argmax_{z∈Adj26(y)} ( ((z−y)·V(y)) / (|(z−y)||V(y)|) ) = x then
                    S ← S ∪ {x}
                    Q ← Q ∪ {y}
                    stopGrowing ← false
                end if
            end for
        end if
    end for
end function
```

The texture system is optimized for fetching and caching data from 2D and 3D textures [37, 1] (see Fig. 1 for an overview of the memory hierarchy on the GPU). The fetch unit of the texture system is also able to perform interpolation and data type conversion in hardware.

Since most of the calculations in this implementation involves the processing of voxels, the implementation can be accelerated considerably by storing the volumes as 3D textures and using the texture system. This increases the speed of fetching data and trilinear interpolation which is used in the TDF calculation when sampling arbitrary points on a circle.

In this implementation, textures has been used for almost all 3D and 2D structures, such as the vector field $\mathbf{V}$, TDF and segmentation.

NVIDIA's OpenCL implementation does not support writing to 3D textures in a kernel. Thus for NVIDIA GPUs, the results has to be written to a regular buffer first and then copied to a texture. Still, writing to 3D textures is possible with CUDA.

Memory access latency can also be improved by reducing the number of bytes transferred from global memory to the chip. The most common way to store a floating point number on a computer is by using 32 bits with the IEEE 754 standard. However, most GPUs also support a texture storage format called 16-bit normalized integer. With this format, the data is stored as 16-bit integers (shorts) in textures. However, when it is requested, the texture fetch unit converts the 16-bit integer to a 32-bit floating point number with a normalized range from -1.0 to 1.0 or 0.0 to 1.0. This reduces accuracy, and may not be sufficient for all applica-

tions. However, it was found to be sufficient for this application (see result section). This storage format also halves the global memory usage, thus allowing much larger volumes to fit in the limited GPU memory. In our recent work on optimizing GVF for GPU execution [42], it was discovered that using textures and the 16-bit format could make the parallel execution a lot faster, depending on the size of the dataset being processed. In this implementation, the 16-bit normalized integer format is used for the dataset, vector fields and TDF result.

2.6.2 Stream compaction

After finding the candidate centerpoints, we only want to process these points in the next centerpoint filtering step. This can be done by launching a kernel for every voxel in the volume and have an if statement checking whether the voxel is a candidate centerpoint. However, this can be very inefficient on a GPU. As explained in the introduction, the functional units on the GPU are grouped together and share a control unit. This means that the functional units in a group have to execute the same instructions in each clock cycle. To ensure that the correct result is generated by if statements, the GPU will use masking techniques. Nevertheless, such an if statement may not reduce the processing time as it would if it was executed sequentially on a CPU. On a GPU, it might even increase the processing time due to the need of masking techniques to ensure correct results. This is a common problem in GPU computing and one solution is a method called stream compaction. Stream compaction removes voxels that should not be processed from the volume so that the kernel is only run for the valid voxels, thus no if statement is needed. Stream compaction can be done on the GPU with logarithmic time complexity. Two methods for performing stream compaction is parallel prefix sum (see Billeter et al. [11] for an overview) and Histogram Pyramids by Ziegler et al. [49]. In this work, Histogram Pyramids has been used due to the fact that this data structure has shown to be better in some applications by exploiting the GPU's texture system for faster memory access. The original implementation by Ziegler et al. [49] was for 2D. However, in our previous work [43], we presented a 3D version of this stream compaction algorithm which also reduced the memory usage for this data structure.

The Histogram Pyramid stream compaction method has been used in three places of this implementation. All in the centerline extraction step. The 3D Histogram Pyramid is used after the candidate centerpoint step and filter centerpoints step. A 2D Histogram Pyramid is used after the link centerpoints step, where each link is stored in an adjacency matrix on the GPU.

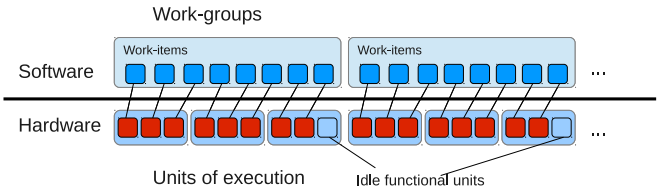

Fig. 5 Grouping with work-group size of 8 work-items and unit of execution size of 3. As 8 is not a multiple of 3, there will be idle functional units for each work-group that is scheduled. This leads to an inefficient use of the GPU.

2.6.3 Work-group size

Work-items, also called threads, are instances of a kernel and are executed on the GPU in groups. AMD calls these units of execution *wavefronts*, while NVIDIA calls them *warps*. The units are executed atomically and has, at the time of writing, the size of 32 and 64 work-items for NVIDIA and AMD GPUs respectively. The work-items are also grouped together at a higher level in software. These groups are called work-groups in the OpenCL terminology (in CUDA they are referred to as thread blocks). If the number of work-items in a work-group is not a multiple of the unit of execution size, some of the GPUs' functional units will be idle for each work-group that is executed as shown in Fig. 5. Thus, the work-group sizes can greatly affect performance and optimal size can vary a lot from device to device. There is a maximum number of work-items that can exists in one work-group. This limit is on AMD GPUs currently 256 and on most NVIDIA GPUs it is 1024. Also, the total number of work-items in one dimension has to be dividable by the size of the work-group in that dimension. So, for a volume of size 400 in the x direction, the work-group can have the size 2 or 4 in the same direction, but not 3, because 400 is not dividable by 3.

For most of the GPUs used on this implementation a work-group size of 4x4x4 was used. One exception is the new Kepler GPUs from NVIDIA where a work-group of 16x8x8 was found to be much better. The 4x4x4 work-group size gives a total of 64 work-items in each work-group. To make sure that the cropped volume is dividable by 4 in each direction, the size of the cropping is increased until the new size is dividable by 4.

2.7 Evaluation

In this section, the evaluation of the proposed GPU method is described.

2.7.1 Comparison with other methods

The method in this paper was compared in terms of speed and quality with other commonly used segmentation and

centerline extraction algorithms. Blood vessels from the MR Angio, Doppler Ultrasound and synthetic datasets were segmented using thresholding after performing Gaussian blur. As thresholding is unsuitable for segmenting airways, an implementation of region growing, similar to the conservative region growing used in Graham et al. [16], was used instead. This region growing methods starts by automatically finding a seed point inside *trachea*. This is done by looking for a dark circular region in the middle of one of the upper slices. After a seed has been found, the dataset is filtered with a Gaussian mask with $\sigma = 0.5$ voxels and the intensities are capped at -500 HU as no airways have intensities above this threshold. Next, a region growing procedure with segmentation leakage detection is used. The region growing is performed several times with increasing threshold starting with the intensity of the seed. For each iteration, the volume size is measured. If the volume size increases with more than 20 000 voxels in one iteration a segmentation leakage has most likely occurred and the previous threshold is used. Finally, a morphological closing is performed to remove any holes inside the segmentation.

The proposed GPU implementation can be used together with both the PCE algorithm and the ridge traversal algorithm for the centerline extraction step. Thus, with the serial ridge traversal algorithm a hybrid solution is used where all steps except the centerline extraction step is run on the GPU.

For the centerline extraction, the proposed GPU method is evaluated with both the proposed PCE centerline algorithm and the ridge traversal algorithm and compared to an ITK filter by Homann [22] based on the skeletonization algorithm by Lee et al. [28]. This skeletonization method performs iterative thinning of a segmented volume. Note that the implementation by Homann [22] does not exploit parallelism.

2.7.2 Qualitative analysis

To show the general applicability of the method, clinical images from three different modalities and two different organs were used:

1. Computer Tomography scans of the lungs (Airways, 12 datasets)
2. Magnetic Resonance images of the brain (Blood vessels, 4 datasets)
3. 3D Ultrasound Doppler images of the brain (Blood vessels, 7 datasets)

The study was approved by the local ethics committee, and the patients gave informed consent prior to the procedure. For each modality, several datasets were processed using the proposed GPU implementation together with the PCE and the ridge traversal centerline algorithms and region growing / thresholding together with skeletonization.

Note that for each modality the same parameters were used, except for a small set of modality dependent parameters such as blur and radius (see Table 1).

2.7.3 Speed and memory usage

The speed of the method was measured on all the clinical datasets using three different GPUs from both AMD and NVIDIA. Two high-end GPUs with a peak performance of around 4 tera floating point operations per second (TFLOPS) (AMD HD7970 and NVIDIA Tesla K20). And one GPU of the previous generation with a peak performance of about 1 TFLOPS (NVIDIA Tesla C2070). The implementation was run using both 16-bit normalized integers and 32-bit floating point vectors to see how the two different data types affected the speed. The proposed method was also run on an Intel i7-3770 CPU (4 cores, 3.4 GHz) with 16 GB memory to show the speedup of using a GPU versus a multi-core CPU. This was also done to demonstrate that the proposed implementation can be run in parallel on a multi-core CPU with no modification.

For comparison, runtime measurements for region growing, thresholding and skeletonization were performed for each modality using an Intel i7-3770 CPU with 4 cores running at 3.4 GHz. Parts of the region growing and thresholding methods were parallelized using OpenMP.

As explained earlier, the memory available on GPUs is limited. Thus it is important to keep the memory usage as low as possible. In this paper, a cropping procedure and a 16-bit normalized integer data format was used to reduce the memory usage on the GPU. To show the effect of the cropping procedure, the average dataset size and peak memory usage before and after cropping was measured on several datasets from different modalities. Peak memory usage occurs in the Gradient Vector Flow step. In this step, 3 vector fields with 3 components, each of the same size as the dataset are needed. For an uncropped volume of size 512x512x800 and 32-bit floats this amounts to $3*3*4*512*512*800$ bytes = 7200 MB. When using 16-bit normalized integers the memory usage is halved.

2.7.4 Quantitative analysis

The quality of the extracted centerlines and the segmentation were measured using realistic synthetic vascular tree volumes and their ground truth segmentation and centerlines. These synthetic volumes and their ground truth data were created using the VascuSynth software by Hamarneh and Jassi [17,23]. One of these synthetic volumes is depicted in Fig. 6. Three generated datasets were used. Each with a different amount of Gaussian additive noise. This was done to show how well the different methods performs with increasing amounts of noise.

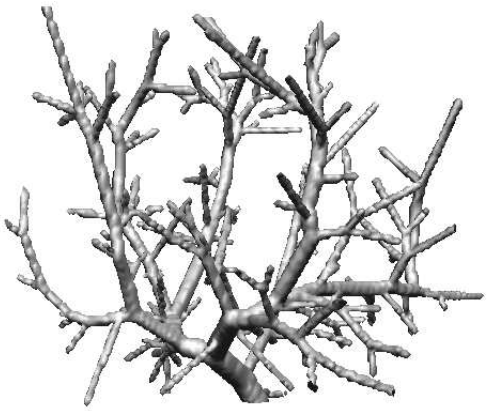

Fig. 6 Synthetic vascular image created using the VascuSynth software by Hamarneh and Jassi [17,23].

Each discrete point of the centerline is called a centerpoint. The accuracy of the centerline was measured using the Hausdorff distance measure which is the average distance from each centerpoint of the extracted centerline to the closest point on the ground truth centerline. To estimate how much of the vascular tree was extracted, each extracted point marks all ground truth centerpoints within a radius of 4 voxels as detected. The total percentage extracted is then calculated as the number of detected points divided by the total number of ground truth centerpoints. Any extracted centerpoint that was farther away than 4 voxels from a grouth truth centerpoint was marked as invalid. The parameters for the amount of Gaussian blur and V_{max} were adjusted for each dataset and centerline method so that no extracted centerpoints were marked as invalid. Precision and recall for the segmentation is calculated by comparing each voxel of the segmentation result to the ground truth.

The quantitative analysis was performed using the proposed GPU implementation with both PCE and ridge traversal and thresholding+skeletonization together with 16-bit normalized integers and 32-bit floating point numbers.

3 Results

3.1 Qualitative analysis

Figures 7, 8 and 9 show results for each method on each modality. Also, to further show the general applicability of the method, extracted vessels from liver and lung is included in Fig. 10. These results indicate that the method is able to extract tubular structures from several modalities and organs with comparable quality by changing only a few parameters (see Table 1).

Parameter	CT Airways	MR Vessels	US Vessels
I_{min}	-1024	100	50
I_{max}	-400	300	200
σ_{small}	0.5	1.0	2.0
σ_{large}	1.0	1.0	3.0
V_{max}	0.3	0.1	0.1
r_{min}	0.5	0.5	1.5
r_{max}	25	8	7
L_{min}	128	10	0

Table 1 A list of modality dependent parameters and the values used for each of the datasets.

3.2 Speed and memory usage

The speed measurements of our GPU implementation with the proposed centerline extraction method and the ridge traversal algorithm is collected in Table 2. These results show that using 16-bit normalized integers is faster than 32-bit on AMD GPUs, and opposite on NVIDIA GPUs.

Table 3 contains speed measurements of the non-GPU methods: region growing, thresholding and skeletonization. Comparing the runtime of Table 2 and 3 reveals that the GPU methods are much faster than the simple serial segmentation and skeletonization methods.

Table 4 shows the average memory usage for all the clinical datasets, both with and without cropping and the 16-bit data type. From these results it is evident that the memory usage is significantly reduced when cropping and 16-bit normalized integers are used.

3.3 Quantitative analysis

Table 5 contains the results of the quantitative analysis described in 2.7.4. From these results it is clear that using the 16-bit normalized integer format does not affect the quality compared to using the standard 32-bit floating point numbers. The same applies to the clinical datasets.

Furthermore, thresholding is able to extract more from the synthetic datasets for noise levels 0.1 and 0.2. However, for noise level 0.3, the proposed PCE algorithm is able to extract almost 10% more than the thresholding and skeletonization technique and the ridge traversal algorithm.

4 Discussion

4.1 Qualitative analysis

The results of the clinical datasets (Fig. 7, 8 and 9) indicate that the quality of the segmentation and centerlines are quite comparable with some small differences. However, if the segmented tubular structure is very irregular or has holes, skeletonization will create poor centerlines as can be seen in Fig. 9. The PCE and ridge traversal algorithms however, do

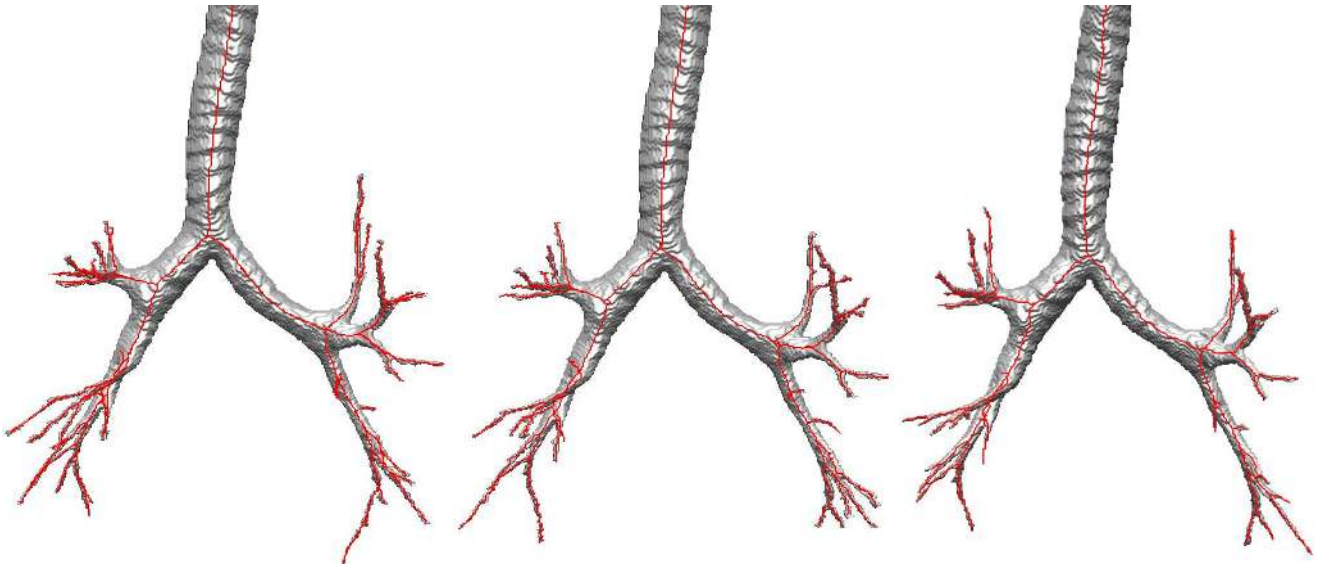

Fig. 7 Results for a CT image of the lungs. **Left:** Proposed GPU method + proposed PCE algorithm. **Middle:** Proposed GPU method + ridge traversal algorithm. **Right:** Region growing with skeletonization

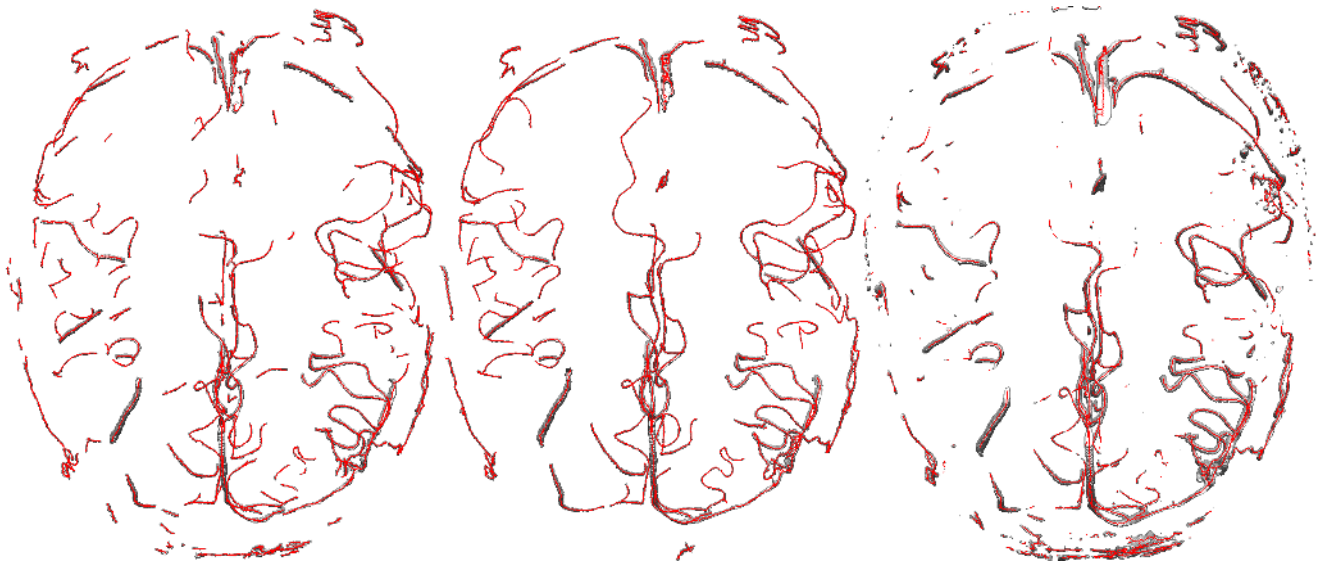

Fig. 8 Results for an MR Angio image of the brain. **Left:** Proposed GPU method + proposed PCE algorithm. **Middle:** Proposed GPU method + ridge traversal algorithm. **Right:** Thresholding with skeletonization

Method	**Datasets**	**AMD HD7970** 16-bit / 32-bit (secs)	**NVIDIA Tesla K20** 16-bit / 32-bit (secs)	**NVIDIA Tesla C2070** 16-bit / 32-bit (secs)	**Intel i7-3770 CPU** 32-bit (secs)
Proposed GPU implementation + Proposed PCE	CT Airways (12)	4.7 / 6.9	21.9 / 13.4	40.9 / 19.2	177.1
	MR Vessels (4)	4.6 / 6.6	28.7 / 16.4	44.9 / 26.6	200.7
	US Vessels (7)	2.7 / 3.8	13.0 / 7.1	24.1 / 14.8	134.4
Proposed GPU implementation + Ridge traversal	CT Airways (12)	5.8 / 8.3	22.3 / 13.9	37.3 / 19.3	175.9
	MR Vessels (4)	6.3 / 8.5	29.7 / 17.5	45.3 / 27.3	200.5
	US Vessels (7)	3.4 / 4.7	13.3 / 7.4	24.1 / 15.0	141.5

Table 2 Average runtime of 10 runs using the proposed GPU implementation together with the proposed parallel centerline algorithm and the ridge traversal centerline algorithm on different datasets and devices. The first three devices (HD7970, K20, C2070) are GPUs while the last device (i7-3770) is a multi-core CPU.

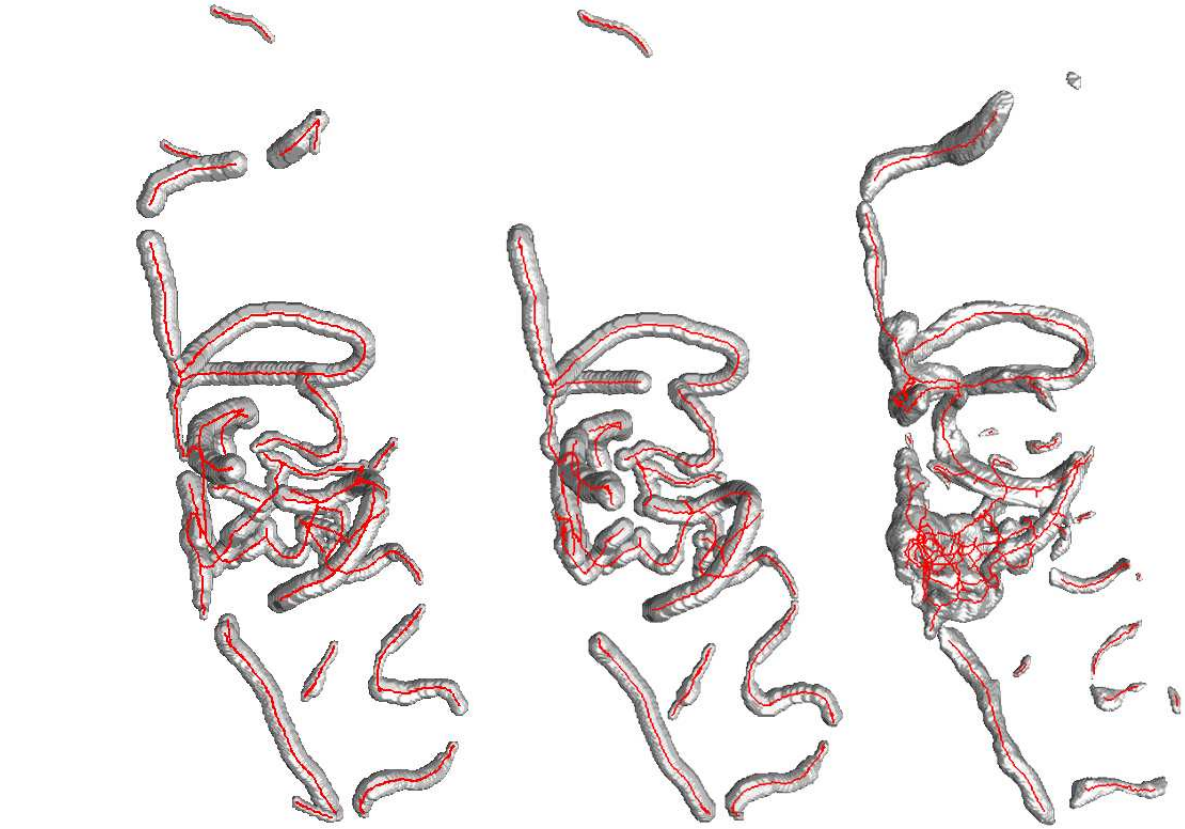

Fig. 9 Results for a 3D Ultrasound Doppler image of vessels in the brain. **Left:** Proposed GPU method + proposed PCE algorithm. **Middle:** Proposed GPU method + ridge traversal algorithm. **Right:** Thresholding with skeletonization

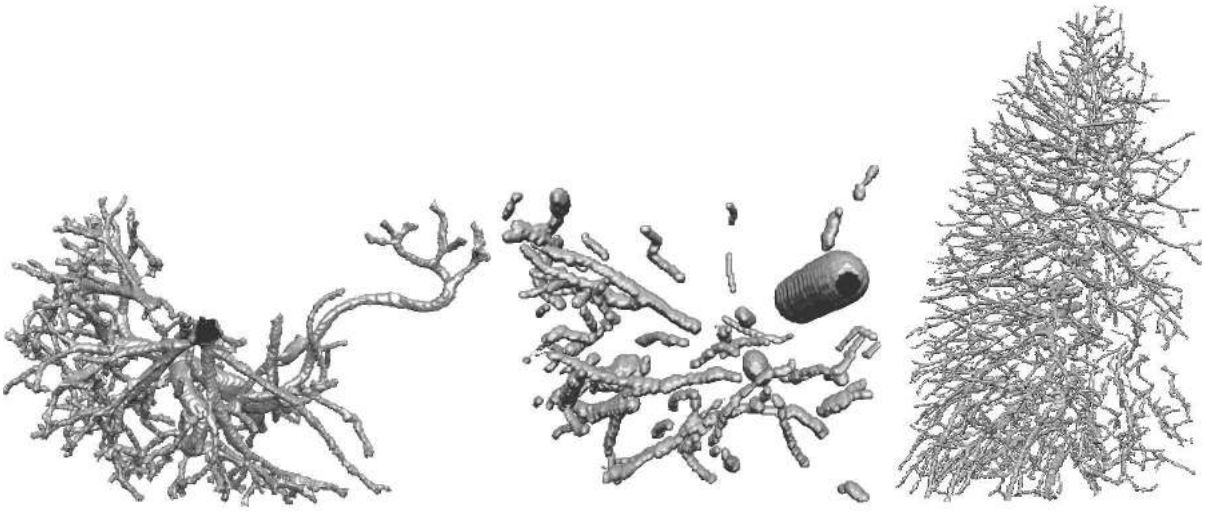

Fig. 10 Segmentation result from other organs using proposed GPU method. From left to right: Vessels of liver from CT, vessels of liver from MR and vessels of one lung from CT.

Segmentation and centerline method	Datasets	Avg. runtime (seconds)
Region Growing + Skeletonization	CT Airways (12)	158
Thresholding + Skeletonization	MR Vessels (4)	77
Thresholding + Skeletonization	US Vessels (7)	33

Table 3 Average runtime of 10 runs using region growing, thresholding and skeletonization/thinning on different modalities.

Datasets	**Avg. original size**	**Avg. percentage removed**	**Avg. peak memory usage without cropping (MB)** 16-bit / 32-bit	**Avg. peak memory usage with cropping (MB)** 16-bit / 32-bit
CT Airways (12)	512x512x704	76%	3169 / 6339	762 / 1524
MR Vessels (4)	628x640x132	23%	2826 / 5652	793 / 1586
US Vessels (7)	272x288x437	31%	1223 / 2445	417 / 834

Table 4 Memory usage and effect of cropping

Dataset	**Noise(σ)**	**Method**	**Avg. centerline error (voxels)** 16-bit / 32-bit	**Extracted centerpoints (%)** 16-bit / 32-bit	**Segmentation recall** 16-bit / 32-bit	**Segmentation precision** 16-bit / 32-bit
Dataset 1	0.1	Proposed GPU method + PCE	0.57 / 0.58	95.6 / 95.6	0.79 / 0.79	0.84 / 0.84
		Proposed GPU method + Ridge traversal	0.35 / 0.35	92.9 / 92.9	0.78 / 0.78	0.84 / 0.84
		Thresholding + Skeletonization	- / 0.34	- / 98.8	- / 0.70	- / 0.99
Dataset 2	0.2	Proposed GPU method + PCE	0.60 / 0.59	80.9 / 80.8	0.57 / 0.57	0.83 / 0.83
		Proposed GPU method + Ridge traversal	0.31 / 0.31	76.1 / 76.1	0.56 / 0.56	0.86 / 0.86
		Thresholding + Skeletonization	- / 0.36	- / 82.1	- / 0.67	- / 0.89
Dataset 3	0.3	Proposed GPU method + PCE	0.65 / 0.65	54.4 / 54.4	0.36 / 0.36	0.79 / 0.79
		Proposed GPU method + Ridge traversal	0.31 / 0.31	42.4 / 42.4	0.28 / 0.28	0.90 / 0.90
		Thresholding + Skeletonization	- / 0.47	- / 45.6	- / 0.47	- / 0.74

Table 5 Performance on three synthetic dataset created with the VascuSynth software (Hamarneh and Jassi [17,23]). For each line, the first value is acquired using 16-bit normalized integers and the second using 32-bit floats.

not suffer from this problem as the centerline extraction is not based on the segmentation result.

There are several examples in the literature of methods that claim to be robust enough to segment and extract centerlines of tubular structures of different types (e.g. vessels and airways), organs and modalities. Some examples are Bauer et al. [3–8], Krissian et al. [26], Aylward et al. [2], Benmansour et al. [10], Li et al. [30], Behrens et al. [9], Cohen et al. [12], Lorigo and Faugeras [33] and Spuhler et al. [44]. However, most of these present results only for a few datasets of one or two organs/modalities. The PhD thesis of Bauer and related articles [3–8] is one exception that present results for several different organs (e.g. lung, heart and liver), however only from CT. Although their approach is similar to the approach in this paper, Bauer et al. use different methods to perform the major steps (tube detection, centerline extraction and segmentation) for each organ. In this paper, results from several organs (e.g. lung, brain and liver), modalities (e.g. CT, MR and Ultrasound) and structures (e.g. vessels and airways) are presented and use the same method for all the major steps. In addition, the method presented in this paper is open source and very fast.

4.2 Speed and memory usage

The proposed GPU implementation is slightly slower using 1-2 seconds more when used with the ridge traversal centerline extraction method than PCE on the two fastest GPUs, the AMD HD7970 and the NVIDIA Tesla K20. However, for the slower GPU, the proposed GPU implementation with ridge traversal is just as fast or even faster. Since this GPU have a peak performance of about one fourth to that of the HD7970 and K20 GPUs, the parallel computation cost of PCE on this slower device is most likely higher than the ridge traversal computation plus the data transfer time.

It is clear from the results that using 16-bit normalized integers instead of 32-bit floats for the vector fields is faster on AMD GPUs, and slower on NVIDIA GPUs. This is due to the fact that NVIDIA's OpenCL implementation does not support writing directly to 3D textures. Because of this restriction, buffers have to be used in the most computationally expensive step, Gradient Vector Flow. This means no 3D cache optimization and hardware data type conversion. Both of which can increase performance.

The runtime of the proposed GPU implementation on a multi-core Intel CPU is several minutes compared to a few seconds on the high-end GPUs. This illustrates the huge speedup gained from running tube detection and segmentation on the GPU.

Skeletonization is the most time-consuming step of the serial methods and is mainly dependent on the thickness of the tubular structures. This is evident in the long execution time of over 2 minutes when processing the airway datasets. Nevertheless, the skeletonization implementation used in this comparison does not exploit parallelism.

Helmberger et al. [21] noted that it is difficult to process a large CT scan due to the limited memory on the GPU. They solved this challenge by decomposing the volume into overlapping sub-volumes that are processed sequentially on the GPU. However, this takes more time and they reported runtime of several minutes. In this paper, the memory limit is avoided by performing cropping and using a 16-bit normalized integer data format. Table 4 shows that the cropping algorithm is able to discard a large portion of the total

input volume. This reduces memory usage significantly and without it, no GPU at the present time would have enough memory to perform the entire calculation in one step for large medical images. Using 16-bit for storage also halves the memory usage allowing larger volumes to be processed entirely on the GPU. On average, the peak memory usage is below 1 GB when cropping and 16-bit data types are used, which is below the memory limit of most modern GPUs.

4.3 Quantitative analysis

The average centerline error is worse for the proposed PCE algorithm than the ridge traversal and skeletonization methods. This increased centerline error is due to the fact that the PCE algorithm creates straight lines between centerpoints. However, it is below 0.7 voxels which we argue is not problematic for most applications and this approximation enables the proposed PCE algorithm to extract over 10% more of the synthetic vascular tree compared to the ridge traversal algorithm for large noise levels (0.3).

Thresholding assumes that all voxels with an intensity above some threshold is part of the tubular structures. This assumption is correct for these synthetic datasets and is thus able to extract more for noise levels 0.1 and 0.2. However, this assumption is usually never correct for a clinical dataset and especially not if the noise level is high. This is evident with noise level 0.3 and in the MR Angio modality in Fig. 8 where the segmentation contains some noise and parts of the cranium.

5 Conclusion

In this article, a fast and generic method that can extract tubular structures such as blood vessels and airways from images of different modalities (CT, MR and US) and organs (brain, lungs and liver) was presented. This was achieved by utilizing the computational power of modern Graphic Processing Units. The method was compared to other methods such as region growing, thresholding, skeletonization by thinning and ridge traversal. Results from both synthetic and clinical datasets from three different modalities (CT, MR and US) was presented. The results show that the method is able to extract airways and vessels in 3-5 seconds on a modern GPU. These near real-time speeds can be beneficial in reducing processing time in image guided surgery applications such as bronchoscopy, laparoscopy and neurosurgery. Although faster and more general than other methods, the quality of the centerline and segmentation was found to be comparable for all the methods.

Acknowledgements Thank you to the people of the Heterogeneous and Parallel Computing Lab at NTNU for all their assistance and St. Olav's University Hospital for the datasets. The authors would also like to convey thanks to NTNU and NVIDIA's CUDA Research Center Program for their hardware contributions to the HPC Lab. Without their continued support this project would not have been possible.

Conflict of interest Erik Smistad, Anne C. Elster and Frank Lindseth declare that they have no conflict of interest.

References

1. AMD. AMD Accelerated Parallel Processing OpenCL Programming Guide. Technical Report December, 2012. http://developer.amd.com/download/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf - accessed 4. July 2013.
2. S. R. Aylward and E. Bullitt. Initialization, noise, singularities, and scale in height ridge traversal for tubular object centerline extraction. *IEEE transactions on medical imaging*, 21(2):61–75, Feb. 2002.
3. C. Bauer. *Segmentation of 3D Tubular Tree Structures in Medical Images*. PhD thesis, Graz University of Technology, 2010.
4. C. Bauer and H. Bischof. A novel approach for detection of tubular objects and its application to medical image analysis. In *Proceedings of the 30th DAGM Symposium on Pattern Recognition*, pages 163–172. Springer, 2008.
5. C. Bauer and H. Bischof. Edge based tube detection for coronary artery centerline extraction. *The Insight Journal*, 2008.
6. C. Bauer and H. Bischof. Extracting curve skeletons from gray value images for virtual endoscopy. In *Proceedings of the 4th International Workshop on Medical Imaging and Augmented Reality*, pages 393–402. Springer, 2008.
7. C. Bauer, H. Bischof, and R. Beichel. Segmentation of airways based on gradient vector flow. In *Proceedings of the 2nd International Workshop on Pulmonary Image Analysis. MICCAI*, pages 191–201. Citeseer, 2009.
8. C. Bauer, T. Pock, H. Bischof, and R. Beichel. Airway tree reconstruction based on tube detection. In *Proceedings of the 2nd International Workshop on Pulmonary Image Analysis. MICCAI*, pages 203–214. Citeseer, 2009.
9. T. Behrens, K. Rohr, and H. S. Stiehl. Robust segmentation of tubular structures in 3-D medical images by parametric object detection and tracking. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 33(4):554–61, Jan. 2003.
10. F. Benmansour and L. D. Cohen. Tubular Structure Segmentation Based on Minimal Path Method and Anisotropic Enhancement. *International Journal of Computer Vision*, 92(2):192–210, Mar. 2010.
11. M. Billeter, O. Olsson, and U. Assarsson. Efficient stream compaction on wide SIMD many-core architectures. In *Proceedings of the Conference on High Performance Graphics*, pages 159–166, 2009.
12. L. D. Cohen and T. Deschamps. Segmentation of 3D tubular objects with adaptive front propagation and minimal tree extraction for 3D medical imaging. *Computer methods in biomechanics and biomedical engineering*, 10(4):289–305, Aug. 2007.
13. O. Eidheim, J. Skjermo, and L. Aurdal. Real-time analysis of ultrasound images using GPU. *International Congress Series*, 1281:284–289, May 2005.
14. M. Erdt, M. Raspe, and M. Suehling. Automatic hepatic vessel segmentation using graphics hardware. In *Proceedings of the 4th international workshop on Medical Imaging and Augmented Reality*, pages 403–412, 2008.
15. A. Frangi, W. Niessen, K. Vincken, and M. Viergever. Multiscale vessel enhancement filtering. *Medical Image Computing and Computer-Assisted Intervention*, 1496:130–137, 1998.

16. M. W. Graham, J. D. Gibbs, D. C. Cornish, and W. E. Higgins. Robust 3-D airway tree segmentation for image-guided peripheral bronchoscopy. *IEEE transactions on medical imaging*, 29(4):982–97, Apr. 2010.
17. G. Hamarneh and P. Jassi. VascuSynth: simulating vascular trees for generating volumetric image data with ground-truth segmentation and tree analysis. *Computerized medical imaging and graphics*, 34(8):605–616, Dec. 2010.
18. M. Hassouna and A. Farag. On the extraction of curve skeletons using gradient vector flow. In *IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.
19. K. Hawick, a. Leist, and D. Playne. Parallel graph component labelling with GPUs and CUDA. *Parallel Computing*, 36(12):655–678, Dec. 2010.
20. Z. He and F. Kuester. GPU-Based Active Contour Segmentation Using Gradient Vector Flow. In *Advances in Visual Computing*, pages 191–201, 2006.
21. M. Helmberger, M. Urschler, M. Pienn, Z. Bálint, A. Olschewski, and H. Bischof. Pulmonary Vascular Tree Segmentation from Contrast-Enhanced CT Images. In *Proceedings of the 37th Annual Workshop of the Austrian Association for Pattern Recognition*, pages 1–10, Apr. 2013.
22. H. Homann. Implementation of a 3D thinning algorithm. *The Insight Journal*, 2007.
23. P. Jassi and G. Hamarneh. VascuSynth: Vascular Tree Synthesis Software. *The Insight Journal*, 2011.
24. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, Jan. 1988.
25. C. Kirbas and F. Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36(2):81–121, June 2004.
26. K. Krissian, G. Malandain, and N. Ayache. Model-Based Detection of Tubular Structures in 3D Images. *Computer Vision and Image Understanding*, 80(2):130–171, Nov. 2000.
27. T.-Y. Law and P. A. Heng. Automated extraction of bronchus from 3D CT images of lung based on genetic algorithm and 3D region growing. *Proceedings of SPIE*, 3979:906–916, 2000.
28. T. Lee, R. Kashyap, and C. Chu. Building skeleton models via 3-D medial surface/axis thinning algorithms. *CVGIP: Graphical Model and Image Processing*, 56(6):462–478, 1994.
29. D. Lesage, E. D. Angelini, I. Bloch, and G. Funka-Lea. A review of 3D vessel lumen segmentation techniques: models, features and extraction schemes. *Medical image analysis*, 13(6):819–845, Dec. 2009.
30. H. Li and A. Yezzi. Vessels as 4-D curves: global minimal 4-D paths to extract 3-D tubular surfaces and centerlines. *IEEE transactions on medical imaging*, 26(9):1213–23, Sept. 2007.
31. P. Lo, B. V. Ginneken, J. M. Reinhardt, and M. de Bruijne. Extraction of Airways from CT (EXACT'09). In *Second International Workshop on Pulmonary Image Analysis*, pages 175–189, 2009.
32. P. Lo, J. Sporring, H. Ashraf, J. J. H. Pedersen, and M. de Bruijne. Vessel-guided airway tree segmentation: A voxel classification approach. *Medical image analysis*, 14(4):527–538, Mar. 2010.
33. L. Lorigo and O. Faugeras. Codimension-two geodesic active contours for the segmentation of tubular structures. In *Computer Vision and Pattern Recognition*, pages 444–451, 2000.
34. J. B. A. Maintz and M. A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.
35. R. Malladi, J. Sethian, and B. Vemuri. Shape Modeling with Front Propagation: A Level Set Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, 1995.
36. A. Narayanaswamy, S. Dwarakapuram, C. S. Bjornsson, B. M. Cutler, W. Shain, and B. Roysam. Robust adaptive 3-D segmentation of vessel laminae from fluorescence confocal microscope images and parallel GPU implementation. *IEEE transactions on medical imaging*, 29(3):583–597, Mar. 2010.
37. NVIDIA. OpenCL Best Practices Guide. Technical report, 2010. http://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf - accessed 4. July 2013.
38. I. Reinertsen, F. Lindseth, G. Unsgaard, and D. L. Collins. Clinical validation of vessel-based registration for correction of brain-shift. *Medical image analysis*, 11(6):673–684, Dec. 2007.
39. L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang. A survey of GPU-based medical image computing techniques. *Quantitative Imaging in Medicine and Surgery*, 2(3):188–206, 2012.
40. I. Sluimer, A. Schilham, M. Prokop, and B. van Ginneken. Computer Analysis of Computed Tomography Scans of the Lung: A Survey. *IEEE transactions on medical imaging*, 25(4):385–405, Apr. 2006.
41. E. Smistad, A. C. Elster, and F. Lindseth. GPU-Based Airway Segmentation and Centerline Extraction for Image Guided Bronchoscopy. In *Norsk informatikkonferanse*, pages 129–140. Akademika forlag, 2012.
42. E. Smistad, A. C. Elster, and F. Lindseth. Real-time gradient vector flow on GPUs using OpenCL. *Journal of Real-Time Image Processing*, 2012.
43. E. Smistad, A. C. Elster, and F. Lindseth. Real-Time Surface Extraction and Visualization of Medical Images using OpenCL and GPUs. In *Norsk informatikkonferanse*, pages 141–152. Akademika forlag, 2012.
44. C. Spuhler, M. Harders, and G. Székely. Fast and Robust Extraction of Centerlines in 3D Tubular Structures Using a Scattered-Snakelet Approach. *Proc. SPIE*, 6144, Mar. 2006.
45. B. van Ginneken, W. Baggerman, and E. M. van Rikxoort. Robust segmentation and anatomical labeling of the airway tree from thoracic CT scans. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 11:219–26, Jan. 2008.
46. A. Vasilevskiy and K. Siddiqi. Flux maximizing geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1565–1578, Dec. 2002.
47. C. Xu and J. Prince. Snakes, shapes, and gradient vector flow. *Image Processing, IEEE Transactions on*, 7(3):359–369, 1998.
48. Z. Zheng and R. Zhang. A Fast GVF Snake Algorithm on the GPU. *Research Journal of Applied Sciences, Engineering and Technology*, 4(24):5565–5571, 2012.
49. G. Ziegler, A. Tevs, C. Theobalt, and H. Seidel. On-the-fly point clouds through histogram pyramids. In *Vision, modeling, and visualization 2006: proceedings, November 22-24, 2006, Aachen, Germany*, page 137. IOS Press, 2006.