Parallelized Segmentation of CT-Angiography datasets using CUDA

Bachelor thesis

Author: Daniel Fischl (e0825242)
Supervisor: Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

June 17, 2012

Contents

Abstract

Segmentation of CT-Angiography datasets is an important and difficult task. Several algorithms and approaches have already been invented and implemented to solve this problem. In this work, we present automatic algorithms for the segmentation of these CTA datasets, implemented in CUDA, and evaluate our results regarding speed and error rates. Starting with local approaches like thresholding we proceed to global, object-based algorithms, like region growing and a newly developed algorithm based on dual energy CT scans (DECT), the XOR-Algorithm, presented by Karimov et al.[6] A limitation of using graphics hardware is the restricted amount of memory, which led us to use a slab-based processing approach (see section 5.3). The requirement of this work was a complete GPU implementation. But since not every task is appropriate for parallelizing, it was necessary to use iteratively parallel algorithms. This strategy though introduced speed problems that had to be analysed and were partly solved. This work presents the principle of these GPU methods and compares them to their CPU counterparts. In the end, the quality of each algorithm is analysed and they are compared against each other, in order to find an acceptable completely automatic segmentation algorithm for distinguishing between different types of tissues (e.g. vessels, bones, soft tissue, ...).

1 Introduction

Segmentation and classification of peripheral CT-Angiography data is a very relevant research field in the field of medical image processing. A lot of research work tries to solve this challenging task and find automatic segmentation algorithms which would be applicable for practical usage. Most works focus on finding an optimal solution and not on implementation details like processing speed. So in this work, we are trying to have a look on the possibility of generating some parallel segmentation and classification algorithms on GPU and therefore achieve two major goals:

- Distinguish between bone and vessel tissue by minimizing the error rate
- Develop algorithms completely on GPU and gain processing speed in comparison to the CPU

In the course of this thesis we will have a look on 3 major algorithms: thresholding, a region growing based approach and a newly presented approach, XOR-Segmentation. Thresholding is considered as the simplest segmentation algorithm. The more sophisticated algorithm, the region growing approach, also makes use of several thresholds and uses a classification on a per-object base. The last presented approach, the XOR-Segmentation, makes use of additional information. The additional information is provided in form of a scan of the human body using two energy levels, which delivers information that can be used for classification. Having a lower and a higher energy scans is referred to as dual energy CT (DECT).
For the purpose of developing these algorithms on GPU, we made use of NVIDIA's Compute Unified Device Architecture (CUDA). These GPU versions were tested for performance against their CPU counterparts. For depicting the practical applicability of the algorithms, we have calculated the error rates of each algorithm, based on provided ground truth datasets.

2 Related work

In the field of segmenting CTA datasets, lots of research is going on at the moment since it is a rather newly emerged problem, and several advanced techniques have already been presented. State of the art is presented by Felkel [9], which was a literature review and test of approaches. The threshold-morphological method by Sramek [12] uses thresholding in combination with morphologic operations and connected component labeling to segment the bone tissue. This tissue is masked out, leaving behind a dataset where vessels have the highest density. This dataset is used for further segmentation. Another approach for example is the Wave Vessel tracking Algorithm by Zahlten [10], which is based on region growing in waves and enriched by bifurcation detection and vessel graph generation. Other approaches are based on multiple stages like the data enhancement and vessel tracking by Frangi [11], where a preprocessing step is applied to the dataset to enhance tubular structures, followed by tracking of these structures through the centerline with a minimal cost approach.
The range of work presented in this field is ranging from simple general approaches to more sophisticated algorithms right up to algorithms based on probabilty information. Examples probabilistic approaches are the Probabilistic Atlas for pCTA Data Segmentation and the Probabilistic Atlas Combined with Watershed Transform as presented in Straka [8]. They take additional information for object recognition and classification into account, e.g. location and shape, and generate a probabilistic atlas based on this information.

Nevertheless, no optimal automatic solutions have been fount yet. In our work, the focus lies on implementing algorithms that run on the GPU. Most of the algorithms are derived from the CPU versions presented by Kanitsar [3] and Karimov et al. [6], with the goal to gain processing speed and minimizing the error rates.

3 Datasets

Different segmentation approaches have been applied to Computer Tomography Angiography (CTA) scans of humans provided as volumetric datasets by the AKH Wien and Kaiser-Franz-Josef-Spital. Due to the nature of the scanning protocol and the importance of detailed information within the medical field those datasets have very high resolution. This is necessary to preserve fine-structured information of the human body within the scan, like blood vessels. High quality scans generate large amounts of data, ranging from 0.5 to 2 GB per scan of a human body (e.g. a 512x512x1800 scan having a size of $\sim$1 GB).

A new approach for segmentation and classification presented in this work is based on the information provided by dual energy computer tomography (DECT) data. With DECT two sets of data from the same human are produced. One with lower energy (80 kV) and one with higher energy (140 kV). The reason for this is that the major difference between the two datasets is the response on calcium-containing tissues leading to additional information which can be used for classification.

For comparison reasons and testing, a ground truth, manually segmented and classified, has been provided for a dual energy dataset.

4 Setup and Framework

4.1 AngioVis

AngioVis[1] is a joint scientific project that deals with the visualization of large CTA datasets with the purpose of detecting peripheral arterial occlusive diseases (PAOD). Semi-automatic algorithms are used for processing the peripheral CTA datasets to identify vessels and bones and remove the occluding bone structures. The AngioVis framework provides several ways of visualizing the results and the possibility to adjust the classification manually in case of automatically wrongly classified objects. AngioVis is successfully in use for clinical and research purposes.

The algorithms presented in this thesis have been integrated directly into this framework within the plugin for bone segmentation.

4.2 CUDA

CUDA[2] is a general purpose parallel computing architecture introduced by NVIDIA in November 2006 [1]. CUDA enables the programmer to make use of the GPU's tremendous computing power by outsourcing computationally complex tasks directly onto the GPU and process the data in a parallel way. CUDA provides a C-like high-level programming interface, therefore being very intuitive and due to its parallel nature it is especially profitable for tasks that include processing of volumetric datasets.

In our implementation we use CUDA Version 4.0. All functions that are launched on the GPU as parallel threads will always be declared as *kernelFunctionName*. The speed gains of porting the CPU implementations onto the GPU will be presented at the end of the paper in the section Results (see 7).

4.2.1 Memory arrangement

The datasets described in section 3 are provided as simple arrays that can be accessed by simply calculating an offset using the x, y and z-information

[1] `http://www.angiovis.org/`

[2] `http://www.nvidia.com/object/cuda_home_new.html`

of the voxel to be accessed. This kind of data structure can simply be transferred to the GPU's device memory by allocating linear memory on the GPU [1]. The allocation of this linear memory doesn't require any pre-processing of the CPU data structure and the data on the GPU can be accessed just like an ordinary array on the CPU.

For the presented methods a concurrent access to voxels within the image is mandatory. Thus the memory arrangement, memory access and kernel calls in CUDA have been handled similar to the description in [2]. In this thesis *length* is refereed to as the maximal x extent, *width* as the maximal y extent and *height* as the maximal z extent of the volume. Arranging the kernel calls by assigning a grid of *width*height* blocks with *length* threads inside each block makes a concurrent access to each neighbouring voxel straightforward. The offset for the currently processed voxel within each kernel-thread can be calculated as

$$offset = (blockIdx.y * width + blockIdx.x) * length + threadIdx.x$$

Using this scheme the offsets to access the six neighbouring voxels can then be easily calculated as follows:

$$\begin{aligned} left &= offset - 1 \\ right &= offset + 1 \\ up &= offset - length \\ down &= offset + length \\ front &= offset - width * length \\ back &= offset + width * length \end{aligned}$$

5 Segmentation Issues

This section provides a short overview over the specific character of the CT segmentation tasks and the use of the GPU for solving these tasks.

5.1 Goal

Not every algorithm is suitable for implementation on the GPU. The goal is to parallelize the whole segmentation process and execute it on the GPU. Since the task of segmenting these datasets contains lots of steps that require a per-voxel processing, the subtasks can be designed as parallel algorithms that can be executed by concurrent threads on the GPU. A major speed gain is expected in comparison with the CPU implementation. For exact performance measurements refer to the results in Section 7.

5.2 CTA Data Characteristics

The main problem for segmentation and classification is due to the special nature of the CTA datasets. The classification step distinguishes between bone and vessel tissue. Simple classification usually fails due to several reasons [3]:

- *Density Overlapping:* The densities of low-density bone and marrow overlap the density values of enhanced blood in the vessels. Similarly, the high-density bone values overlap the density values of calcifications within the vessel's walls.
- *Adjacency of bones and vessels:* Vessels touching the bones can also complicate the classification due to non-sufficient resolution (Partial Volume Effects)
- *Special cases:* Cases not considered in the segmentation process, like stents or metal implants

Another difficulty is the fact of large datasets versus limited GPU memory. Most of the datasets are far too large to fit completely onto existing device memories making it impossible to process the volume all at once. This (in addition to some other facts) was the reason for processing the data slabwise as described in Section 5.3.

5.3 Slab-processing

Due to the size, the datasets are not processed at once but as so called *slabs*. A slab consists of several consecutive volume slices, whereas the exact number can be predefined (usually ∼30). The implemented algorithms work on these slabs. It is possible to either run the algorithm on a single slab, or run it on the whole volume. The latter is just a special case where the algorithm is applied sequentially on each slab of the whole volume as shown in *Figure 1*. The data of each slab is copied to the GPU's device memory where the algorithms are applied to it. After finishing, the processed data is transferred from the device memory back to the CPU's host memory.

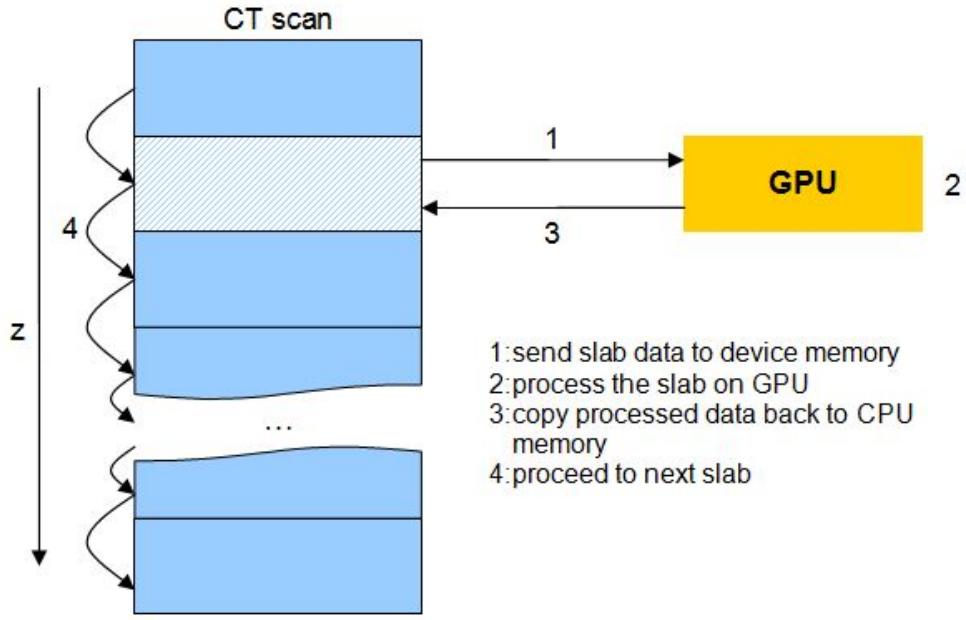

Figure 1: Concept of slab-processing

6 Segmentation

The following section gives a detailed description of used segmentation methods, simple thresholding and two more advanced methods. In particular the latter methods consist of two separate steps: segmentation and classification. The segmentation step is necessary to find potential objects and separate them from the background. The following classification step is then used to distinguish between vessel and bone objects. Each step of the algorithms has been ported to GPU. The kernels execute for every voxel of the currently processed slab.

6.1 Thresholds

The usage of parameters, like thresholds and object size limits (measured in number of voxels), is required for simple, and also for more advanced segmentation algorithms. In the course of our implementation we make use of several parameters that are predefined by trial and error approach and can be adjusted by the user for each slab independently during segmentation. The following enumeration describes the thresholds and limits that were used to segment and classify the objects and are based on the parameters as described in [3], p.54. (The exact values in Hounsfield Units that were used for testing are mentioned in the footnotes):

- *High and Low Energy Thresholds:* [3] the density thresholds used in the case of dual energy XOR segmentation
- *Main Hard Limits and Soft Limit:* [4] thresholds for a general distinction of objects
- *Bone Hard Limit and Soft Limit:* [5] soft limit as minimum required density threshold for a bone object and the hard limit where we assume that the voxel is a bone voxel
- *Bone Size Limit:* [6] minimum required size of an object to be classified as bone tissue

[3] Low Energy: Max=3071, Min=176; High Energy: Max=3071, Min=356
[4] Hard Limit=376, Soft Limit=166
[5] Hard Limit=626, Soft Limit=476
[6] 30

- *Vessel Soft Limit:* [7] minimum required density for vessel object
- *Vessel Size Limits:* [8] minimum required size of object to be classified as vessel
- *Gradient Limit:* [9] maximum limit for the gradient magnitude as criteria for the region growing

6.2 Thresholding

Simple density thresholding [3] is a local (per-voxel) segmentation method commonly used and for comparison the outcome and performance gains of GPU-thresholding will be mentioned here. Due to the overlapping of densities we don't simply use an upper and lower threshold. We make use of thresholding similar to the XOR approach ([6]) which is described in detail in the later section 6.4.3. The advantage of this method is that we can make use of dual energy CT scans even using this simple approach. The following *Algorithm 1* shows the thresholding kernel used in our implementation:

Algorithm 1 Kernel for (XOR)-Thresholding

```
if curVoxel_low ≥ Min AND curVoxel_low ≤ Max then
    if curVoxel_high ≥ Min AND curVoxel_high ≤ Max then
        // both within - Bone
        Mark curVoxel as Bone-Voxel
    else
        // only lower Energy within - Vessel
        Mark curVoxel as Vessel-Voxel
    end if
else
    // no object voxel
end if
```

If the processed dataset is only single energy, then the "low" and "high" energy Voxels used in the kernel both origin from the same scan and therefore have the same value. To "fake" the dual energy solution, the second Min threshold is chosen to be slightly higher than the first one so that the voxel is only classified as bone tissue if it is within both Min-Max ranges.

[7] 156
[8] minimum size: 3, maximum size: 10000
[9] 650

6.3 Region Growing

We use connected component threshold region growing as described in [4], controlled by the gradient magnitude. The implementation is a GPU version based on the segmentation and classification method in [3], pp. 53-58. The basic algorithm consists of the following steps:

1. *Find Seedpoints*
2. *Grow*
3. *Close Holes*
4. *Detect Objects*
5. *Classify Objects*

The algorithm starts with segmentation steps (1 to 4) where it separates objects in the volume from the background. In the last step (5) a classification is being applied to distinguish between bone and vessel tissue. Local thresholding classifies each voxel according to its density value. In order to obtain clearly segmented and classified objects, and not just classified voxels, our region growing and XOR-Segmentation are both object-based approaches. Step 5 is responsible for identifying and classifying the different objects within the CT scan.

6.3.1 Find Seedpoints

An important aspect of this segmentation algorithm is that it runs without any user interaction. Seedpoints are selected by thresholds. In our approach we choose the main hard limit (*MainHL*) as a necessary condition for a voxel to be defined as seedpoints. The simple kernel depicted in *Algorithm 2* is called once and checks for every voxel to be a definite object-voxel which will serve as seedpoints for the next step.

Algorithm 2 Kernel for finding seedpoints used for region growing
kernelFindSeedPoints

if $curVoxel_{density} \geq MainHL$ **then**
    Mark $curVoxel$ as Bone-Voxel
end if

Note: Since the majority of the segmented objects will be bone-objects the complete algorithm will start marking every detected object as bone and perform a re-classification in the classification steps.

6.3.2 Grow

After finding seedpoints, the actual region growing process can start. The parallel kernel depicted in *Algorithm 5* is iteratively called until no more changes occur. This change is monitored by allocating a counter or flag in device global memory that can be set in either of one of the threads. In particular, before each kernel call, the global flag is initialized with $false$ and if any of the threads during kernel execution changes it to $true$ (due to changes still occurring in in the thread) the kernel will be invoked again until no more changes occur in any of the threads. *Algorithm 3* depicts the concept of the iterative calling.

The algorithm expands the seed regions by looking at the neighbours of bone-marked voxels and checking if the growing criteria is fulfilled. The index of the neighbour voxels is calculated as described in Section 4.2.1. The growing criteria in the current implementation is that it is within the general object thresholds main hard limit threshold (*MainHL*) and main soft limit threshold (*MainSL*) supported by checking the gradient magnitude limit threshold (*MainGL*). For the exact definition refer to *Algorithm 4*.

Algorithm 3 Loop that keeps calling the grow-kernel until no changes occur

```
if changed ≤ true then
    kernelRegionGrowing(...)
end if
```

Algorithm 4 Check if, given a density and gradient magnitude value, the voxel can be identified as an part of an object
belongsToObject(density, gradientMagnitude)

```
if density ≥ MainHL then
    return true
else if density ≥ MainSL ∧ gradientMagnitude ≤ MainGL then
    return true
else
    return false
end if
```

Algorithm 5 The iteratively called Region Growing Kernel *kernelRegionGrowing*

if $curVoxel$ not marked as bone **then**
▷ No seed or region voxel - exit thread
  return
else
  for all $neighbourVoxels$ **do**
    $neighbour_{gradiehnt} \leftarrow calculateGradient(neighbourVoxel)$
    if $belongsToObject(neighbour_{density}, neighbour_{gradient})$ **then**
      Mark $neighbourVoxel$ as Bone-Voxel
      $changed \leftarrow true$
    end if
  end for
end if

The computation time for this parallel region growing algorithm depends on the number of seed points found, the structure of the objects and the number of objects in the processed slab. The more seed points, the fewer iteration steps. For snake like shaped objects and for a high number of objects the number of iterations and therefore the computation time will increase accordingly.

6.3.3 Close Holes

The bone marrow inside the bones has a much lower density than the bone itself. Distinction through a threshold is not possible and the gradient magnitude is too high for the voxels to be reached by region growing. Thus the bone marrow would not be classified as bone tissue. Therefore a different approach has been made to ensure correct detection (through CCL) and classification of bone marrow in the later steps. Generally speaking its a region growing applied to the background. At first the outside borders are checked for background voxels (with very low density). Then the growing is initiated and iteratively repeated until no more changes occured. The major difference to the above mentioned algorithm is that the growing process works per slice to prevent a leaking-through through non-relevant small holes somewhere within the slab. After the growing has finished, all Voxels that haven't been reached by the background-growing are marked as bone-voxels. *Figure 2* depicts how a slice might look after region growing and what happens after the "Close Holes" step is applied to the slice.

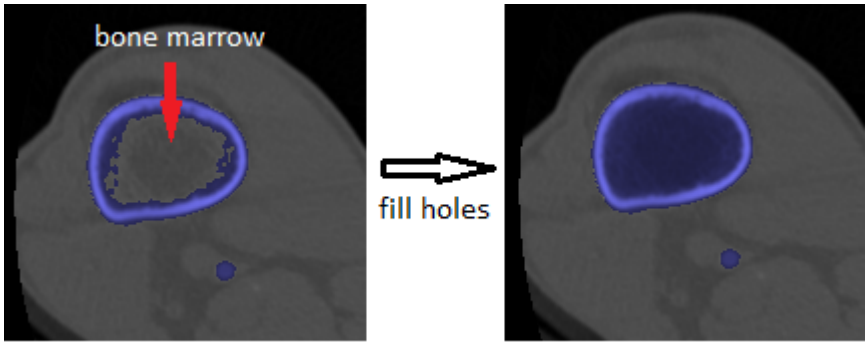

Figure 2: Slice after region growing and before filling holes compared to the same slice after filling holes

6.3.4 Detect Objects

Connected Component Labeling (CCL) has been used to detect all objects and is a necessary step for classification to find objects as a whole and make it possible to calculate object properties. In the course of this thesis we implemented a simplified version of the parallel algorithm described in [5]. The basic idea is to initially give each voxel a unique label (we chose the offset of the voxel which represents the index in the array). Then the kernel is iteratively executed wihin a loop until no more changes occur. The monitoring of changes is done using a global flag (*changed*) as described in Section 6.3.2. For each non-background voxel the neighbours and labels are checked and the minimum label is used as the new current label. This is repeated until no more changes occur (see *Algorithm 6*) leading to a unique label for each connected region.

6.3.5 Classify Objects

Given the labeled objects it is now possible to calculate the properties of each object in parallel using CUDA with its provided atomic functions [1]. The following properties have been calculated:

- Object size
- Average object density

The new information, in combination with the defined soft thresholds (bone and vessel soft limit (*BoneSL, VesselSL*), bone and vessel size limit (*BoneML,*

Algorithm 6 Iteratively called kernel for Connected Component Labeling *kernelCCL*

```
if curVoxel marked as background then
▷ No object voxel - exit thread
    return
else
    minimum ← label_curVoxel
    for all neighbourVoxels not marked as background do
        if label_neighbourVoxel < minimum then
            minimum ← label_neighbourVoxel
        end if
    end for
    if minimum < label_curVoxel then
        label_curVoxel ← minimum
        changed ← true
    end if
end if
```

VesselSizeMin, VesselSizeMax)), is used to further refine classification, as depicted in *Algorithm 7*. Depending on the size of the object and the average density each object is checked and classified as vessel if necessary, otherwise it remains classified as bone. Furthermore very small detected objects are discarded and very large objects with lower average density are left to be bones and not classified as Vessel. The reason for treating large objects this way is is that most of the time the kneecap is classified as vessel due to its lower density.

6.4 XOR-Segmentation

XOR-Segmentation is a new approach for classification based on DECT presented by Karimov et al.[6] The algorithm makes use of the different responses on calcium-containing tissues in lower energy and higher energy scans. In particular lower energy scans allow segmentation of bones and vessels tissues by thresholding, whereas in higher energy scans thresholding can be used to segment bone tissues. *Figure 3* gives an overview of the difference between low and high energy. Combining the resulting masks using an ordinary logical "exclusive or" (XOR) operation leads to a difference mask which is used in the final classification step (see section 6.4.3).

The steps of the XOR-Segmentation are similar to those of the region grow-

Algorithm 7 Classification of the objects due to avg density and object size *classifyObject*

```
for all objects do
    if (avgDensity > BoneHL) OR
     (avgDensity > BoneSL AND size > BoneML) then
        Keep Bone Label
    else
        if avgDensity > VesselSL AND VesselSizeMin < size <
VesselSizeMax then
            Map To Vessel
        else if size < VesselML then
            Map To Nothing
        end if
    end if
end for
```

ing, except a modification in the classification step where the XOR-difference mask is applied. This thesis presents a parallel version of the basic XOR algorithm and is completely implemented on GPU.

1. *Define XOR Mask*
2. *Mark Object Voxels*
3. *Close Holes*
4. *Detect Objects*
5. *Classify Objects*

6.4.1 Define XOR Mask

For computation of the XOR mask we made use of two new thresholds, a maximum and minimum threshold. As mentioned above and seen in *Figure 3*, low and high energy scans respond differently to the same thresholds. This characteristic is used to create the above mentioned difference mask. At first we check if density value of the lower and higher energy scan is within the thresholds. If the lower energy voxel is within the thresholds and the higher energy voxel is also within, then this position within the difference mask gets marked as bone tissue voxel. If only the lower energy scan voxel is within the thresholds and the higher energy is not, then the mask gets marked as

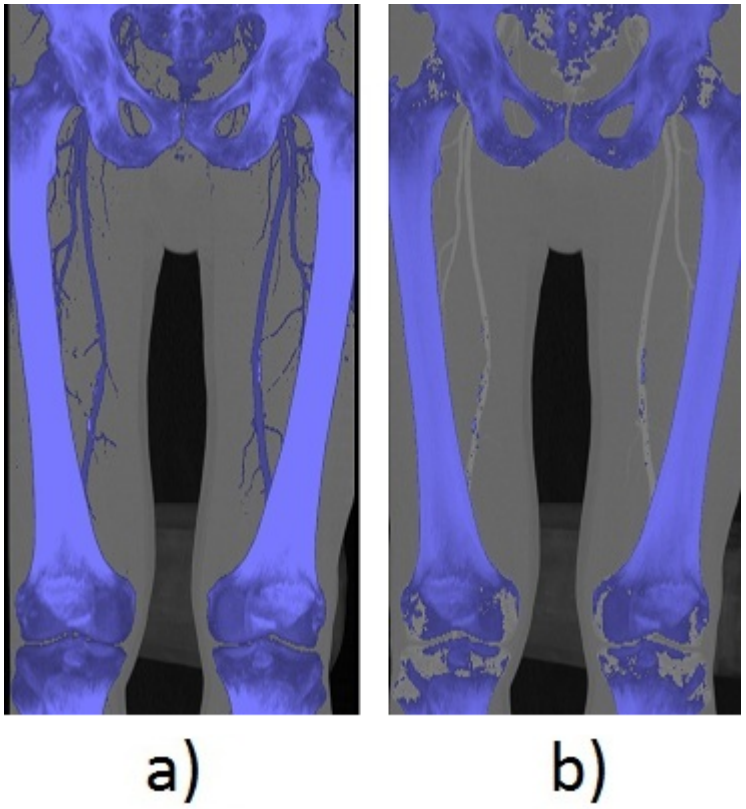

Figure 3: Difference between a) low energy scan thresholding and b) high energy scan thresholding [6]

vessel tissue voxel. The GPU kernel for defining this mask is depicted in *Algorithm 8*. The resulting mask is used for the later classification step (see section 6.4.3).

6.4.2 Mark Object Voxels

In a next step we launch a single kernel-call to mark all voxels in the volume that are potential objects. This is done by using the same function that is used as growing criteria for the region growing (see *Algorithm 4*). The difference is that this criteria is applied to every voxel, and not just voxels that are neighbours of seed or region points. *Algorithm 9* depicts the kernel that is called once.

Algorithm 8 Kernel applying an XOR operation for calculating the difference mask
kernelDefineXORMask

if $curVoxel_{low} \geq Min$ AND $curVoxel_{low} \leq Max$ **then**
  if $curVoxel_{high} \geq Min$ AND $curVoxel_{high} \leq Max$ **then**
    // both within - Bone
    $xorMask_{curVoxel} \leftarrow BONE$
  else
    // only lower Energy within - Vessel
    $xorMask_{curVoxel} \leftarrow VESSEL$
  end if
else
  // no object voxel
  $xorMask_{curVoxel} \leftarrow 0$
end if

Algorithm 9 Check for each voxel if it satisfies the object-condition
kernelMarkObjectVoxels

if $belongsToObject(curVoxel_{density}, curVoxel_{gradient})$ **then**
    Mark $curVoxel$ as Bone-Voxel
end if

6.4.3 Classify Objects

After the detection of objects as described in Sections 6.3.3 and 6.3.4 the final step is the mapping of voxels currently classified as bone tissue to vessel tissue voxels if necessary. As in the classification step of the region growing (see section 6.3.5), the properties can be computed per object. Taking the, by using dual energy, computed XOR difference mask into account, the following object properties are used for reclassifictaion:

- Object size
- Both bone voxels (bone indicator voxels): Voxels that responded to the thresholds in both (high and low energy) scans (i.e. marked as bone in the XOR mask). They are a cue that the object might really be a bone object. The fewer voxels of an object are "bone indicators", the higher the probability of the object being a vessel.

Using these two properties, we use *Algorithm 10* to reclassify the objects. They are classified as vessels if too less voxels within an object (beneath 10% of the object's voxels) are "bone indicators" according to the XOR mask.

Algorithm 10 Classification of the objects due to XOR mask and object size
XORclassifyObject

```
for all objects do
    if avgBoneIndicator < 0.1 AND size < 10000 then
        Map To Vessel
    end if
    if size < VesselML then
        Map To Nothing
    end if
end for
```

7 Results

The results stated in this section have been performed on a computer with a NVIDIA GeForce GT 425M and 4 GB RAM memory. The dataset used for testing the segmentation is a DECT scan of extent 512x512x855 and has a size of 427 MB for each scan (high and low energy scan).

7.1 Speed

Using the GPU as a general computing device has proven to be a very useful tool for parallelizable computationally intensive tasks. Nevertheless, in our implementation, in most cases the GPU versions were not able to outperform their CPU counterparts. *Table 1* shows the mean durations (in seconds) of our segmentation algorithms on GPU compared to the CPU versions of the algorithms when segmenting the whole volume.

	CPU (s)	GPU (s)	diff
Thresholding	3.77	1.22	67.64 %
Thresholding (DECT)	3.77	1.44	61.80 %
Region Growing	91.65	139.82	52.56 %
XOR-Segmentation	129.75	315.34	143.04 %

Table 1: Speed comparisons of GPU and CPU versions

As expected, for the a simple algorithm like Thresholding, a great speed gain of more than 60% has been achieved since a single kernel is able to process each voxel in parallel. Transferring the data from host to device and back again takes up the main part of processing time (e.g. $\sim$0.035 seconds per slab - $\sim$0.95 seconds for the whole volume) and the kernel is executed in a few milliseconds. Knowing this, the lesser speedgain when using DECT information is also logical, since two volumes have to be transferred to GPU.

Having a look at the speeds of the more sophisticated algorithms, we can see that the GPU implementation causes a major speed loss. Therefore we took a closer look on the single steps of the algorithm to find the bottlenecks of our implementation. A look at the processing time of each slab shows that the duration varies from 2 seconds to up to 35 seconds per slab. This is due to the concept of our implementation, which makes a lot of use of iteratively calling the parallel kernels. The number of kernel calls is not fixed for each

slab. For example, the number, and therefore the processing time, depends mainly on the number of objects detected in the slab. This correlation can be seen in *Figure 4*, which shows a plot of the number of objects in a slab versus its processing time.

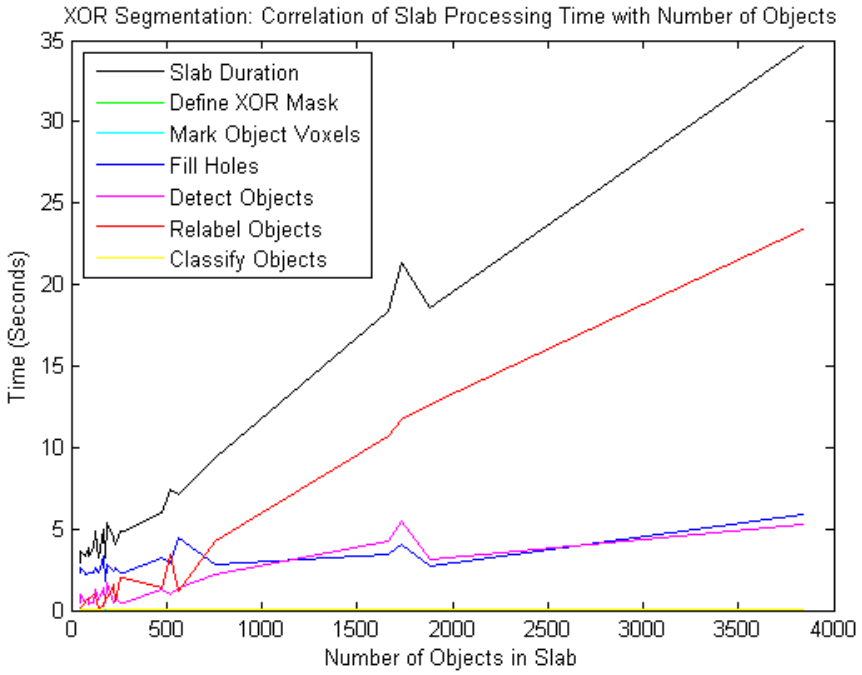

Figure 4: Duration of the single steps of the XOR Segmentation and its correlation to the number of objects detected in the slab using XOR segmentation

The plot shows clearly that a main part of the processing time is taken by the *Relabel Objects* step, which is part of the Connected Component Labeling algorithm implemented. This is due to its linear dependence on the number of objects detected in the slab. In particular, the kernel gets called once for each object detected in the slab. That the GPU implementation is not always slower than the CPU version is depicted in *Figure 5*.

In this plot we can see, that the duration for processing a slab on the CPU is constant and independent of the number of objects in the slab in contrast to the GPU version. But from this plot we can also observe that the GPU implementation is faster than the CPU implementation if only less than $\tilde{5}00$ objects are detected within the slab. So the next step to gain a speed-up

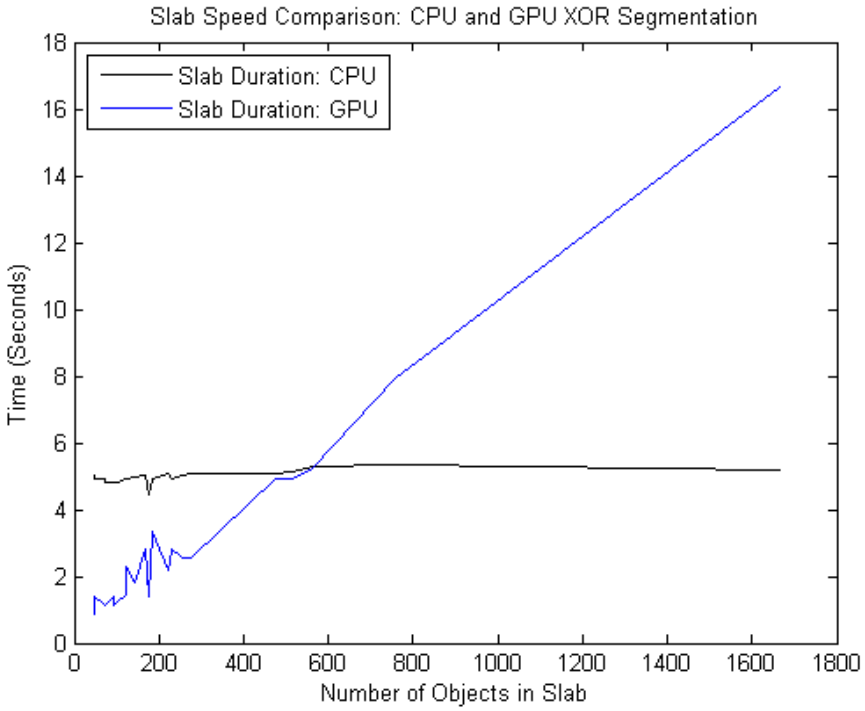

Figure 5: Slab Duration of the CPU XOR Segmentation compared to the GPU version

would be either to decrease the number of unnessecary objects detected in the beginning, or implement a more sophisticated Connected Component Labeling algorithm which is independent of the amount of objects, or implement this part of the algorithm on CPU where we have to consider additional memory transfers from GPU to CPU (and vice versa).

Another potential for gaining some speed is the step *Fill holes* which has been implemented to take care of the bone marrow inside the bones and is implemented using an iterative parallel Region Growing algorithm for the background. That omitting this step doesn't have that much influence on the error rate (see 7.2) can be seen when having a short look at *Figure 6*. Here we can see that our provided Ground Truth segmentation also does not take the bone marrow into account. The drawback in this approach is that within the Bone Marrow there might be objects detected as vessel which can be seen by the red colored voxels within the bone marrow in the rightmost image in *Figure 6*.

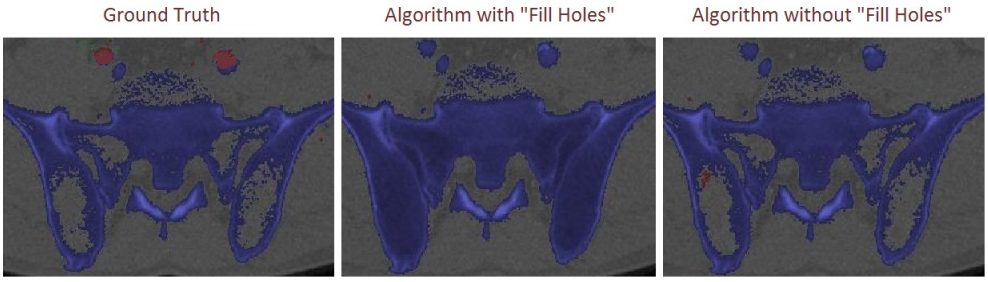

Figure 6: Comparison of the Ground Truth dataset with algorithms using the *Fill Holes* step and algorithms without this step by an example in form of a single slice.

To depict the speed gains we have also implemented both algorithms without this step and the results are summarized in *Table 2*.

	CPU (s)	GPU (s)	diff
Region Growing (w/o. fill holes)	91.65	66.65	27.28 %
XOR-Segmentation (w/o fill holes)	129.75	261.30	101.39 %

Table 2: Speed comparisons of GPU (without filling holes) and CPU versions

Here we achieved a speed gain, since this step took about 2.5 to 3 seconds in each slab. The speed gain for the region growing algorithm sufficed to make it faster than the CPU version. For the XOR-Segmentation we also achieved a gain, but the algorithm is still slower than its CPU counterpart.
For the sake of completeness *Figure 7* shows also a plot of the correlation between processing speed and number of objects for the XOR-Algorithm without hole filling. In this plot we can see the major contribution of the Relabel objects step to processing time which linearly depends on the number of objects, whereas the durations of every other step are mainly constant and independent of the amount of objects.

Note also, that when executing the GPU-based algorithms for the first time, the process of transferring the data (which usually takes a few milliseconds) might take up lots of time (up to 20 seconds), since the memory is internally arranged for the first time and then used multiple times in the following procedure.

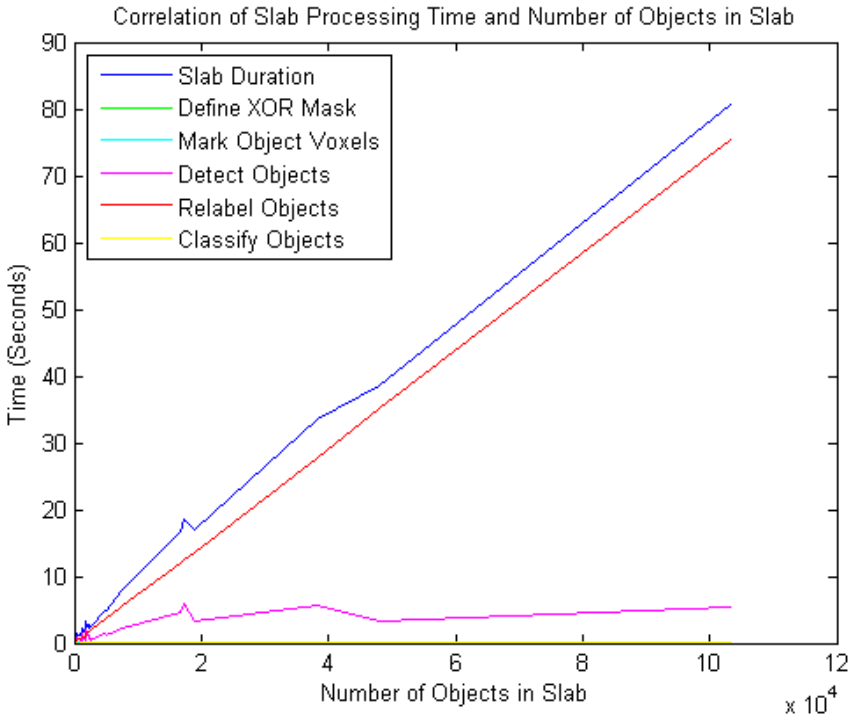

Figure 7: Duration of the single steps of the XOR Segmentation (w/o hole filling) and its correlation to the number of objects detected in the slab using XOR segmentation

7.2 Error rate

This section deals with the error rates based on the Ground Truth segmentation that has been provided and is depicted in *Figure 8*. Consider the following error percentages with caution, since the provided ground truth dataset does not always mark the bone marrow as bone tissue (as mentioned before).

The error rates for the validation have been calculated separately for the bone and vessel class and are given in percentages derived from the confusion matrix as described in [7]. The following error rates have been calculated:

- *False Positive Rate (FPR)*: also referred to as Type I error
- *False Negative Rate (FNR)*: also referred to as Type II error
- *True Positive Rate (TPR)*: 1-FPR

- *True Negative Rate (TNR)*: 1-FNR

The two tables (*Table 3* and *Table 4*) summarize the error rates for each implemented algorithm for the vessel and bone segmentation and are discussed in the following part of this section.

BONE	FPR	FNR	TPR	TNR
Thresholding	0.094 %	48.59 %	51.41 %	99.91 %
Thresholding (DECT)	0.003 %	65.32 %	34.68 %	99.997 %
Region Growing	0.868 %	1.58 %	98.42 %	99.13 %
Region Growing (w/o. fill holes)	0.301 %	2.03 %	97.97 %	99.70 %
XOR-Segmentation	0.774 %	3.17 %	96.83 %	99.23 %
XOR-Segmentation (w/o fill holes)	0.233 %	1.05 %	98.95 %	99.77 %

Table 3: Error rates for class BONE

VESSEL	FPR	FNR	TPR	TNR
Thresholding	1.331 %	67.47 %	32.53 %	98.67 %
Thresholding (DECT)	2.057 %	51.23 %	48.77 %	97.94 %
Region Growing	0.050 %	86.14 %	13.86 %	99.95 %
Region Growing (w/o. fill holes)	0.053 %	86.14 %	13.86 %	99.95 %
XOR-Segmentation	0.152 %	56.97 %	43.03 %	99.85 %
XOR-Segmentation (w/o fill holes)	0.017 %	56.90 %	43.10 %	99.98 %

Table 4: Error rates for class VESSEL

Having a closer look at the bone segmentation, we can see that the Type II errors (FNR) are very high for the simple thresholding algorithms, because they use a per-voxel based classification and missclassify a lot of voxels within and near the bone as vessels, because classification is only based on thresholds and nothing else. That the per-voxel approach is not sufficient enough for our purpose can be seen in the two figures *Figure 9* and *Figure 10*. The slightly higher False Type I errors (FPR) of the ordinary XOR-Segmentation and Region Growing is due to the nature of the ground truth dataset which has not marked the bone marrow as bone. Otherwise, the results for Bone Segmentation look quite promising for the more sophisticated algorithms, with error rates lower than 4%.

Regarding the vessels, the task and error rates are more subtle. The Vessel

structure within the human body is finely branched leading to very small structures which cannot be easily classified. This phenomena can be observed looking at the Type II errors (FNR). These high error rates depict the fine structures that haven't been classified as vessels by our automatic segmentation approaches. Note, that the XOR-Segmentation approach in comparison to the region growing provides better results by also finding thin vessel structures within the body, which can also be seen comparing the image results of region growing (see *Figure 11*) and XOR (see *Figure 12*). Comparing the outcomes of the new XOR-Algorithm with the ground truth shows lots of similarities and results that are very promising for being suitable for practical usage in the field.

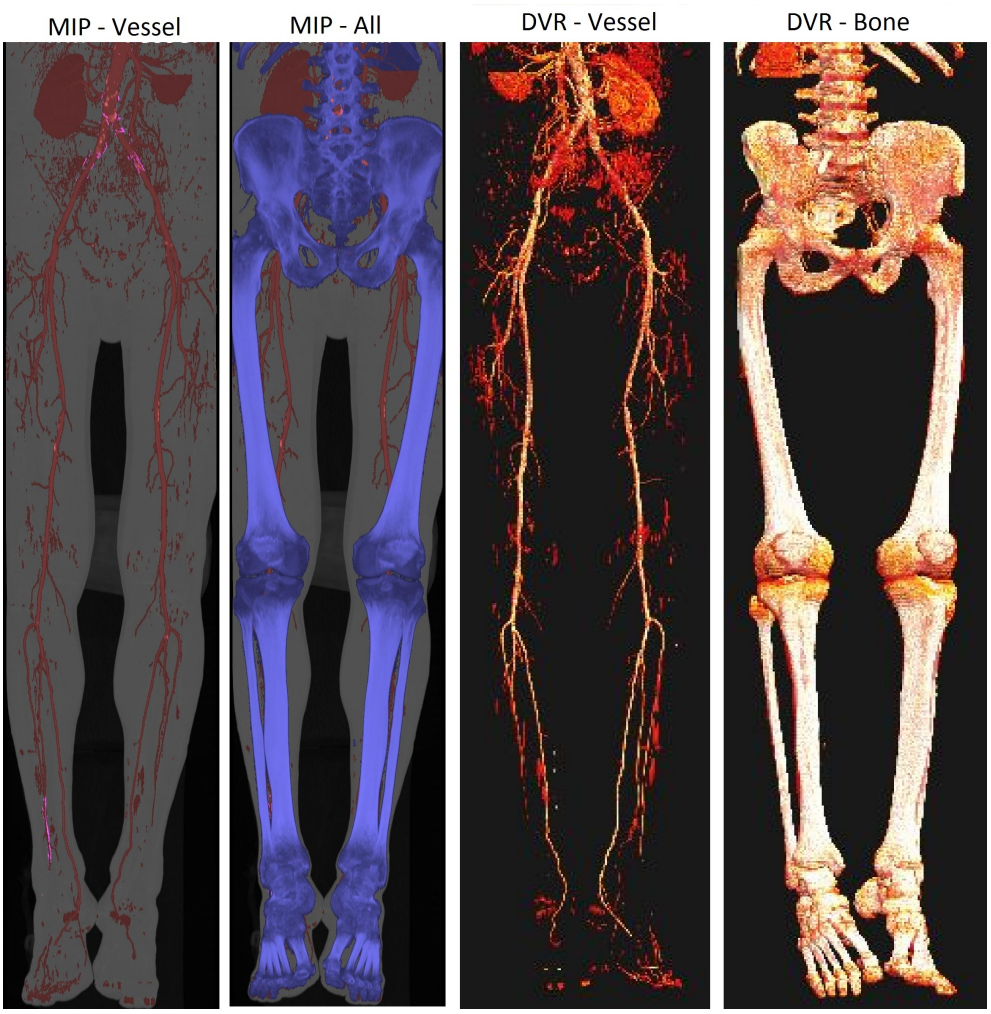

Figure 8: MIP images and direct volume rendered images of the ground truth dataset.

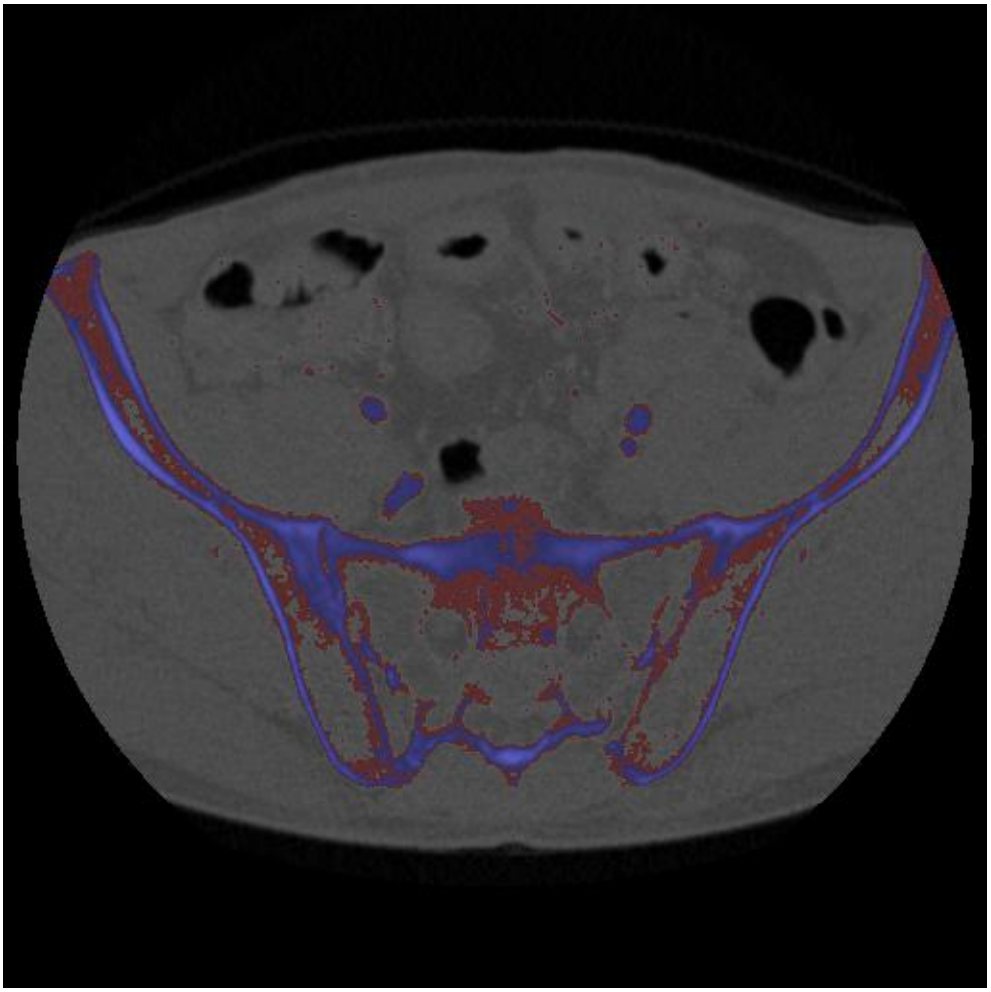

Figure 9: Sample slice segmented with thresholding on a per-voxel basis

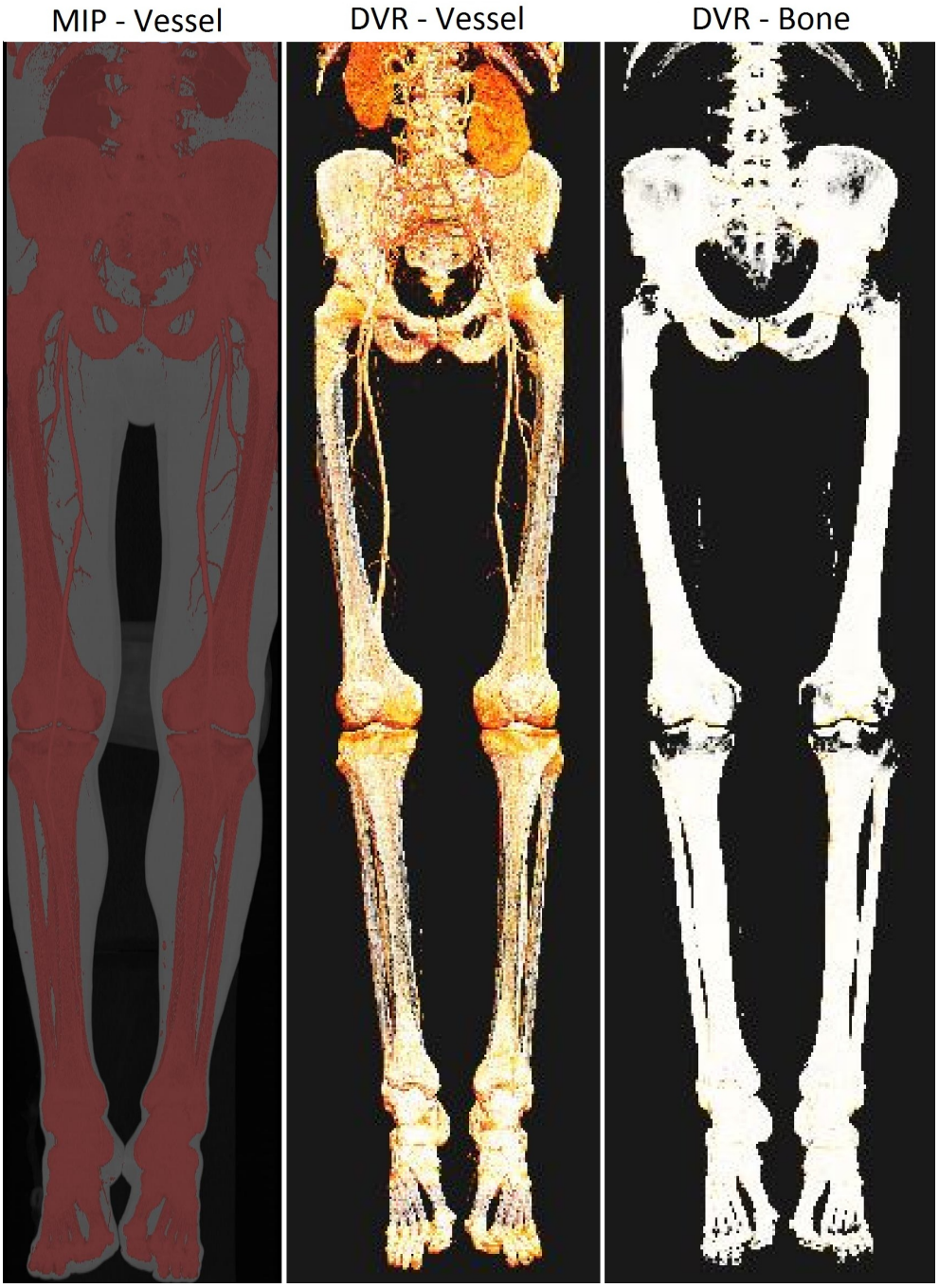

Figure 10: Thresholding: MIP - Vessel image and direct volume rendered image of vessels and bones

Figure 11: Region Growing: MIP - Vessel and MIP - All image and direct volume rendered image of vessels and bones

Figure 12: XOR-Segmentation: MIP - Vessel and MIP - All image and volume rendered image of Vessels and Bones

8 Conclusion

Correct segmentation of CTA datasets has proven to be a very difficult task due to overlapping of densities, close spatial vicinity and other problems. Therefore a complete automatic segmentation algorithm cannot substitute user interaction and intervention in form of manual segmentation or user-drive approaches [8]. Simple algorithms do not suffice at all and more sophisticated algorithms, like region growing, fail in recognizing small vessel structures within the body. However, the new XOR-Algorithm approach, making use of dual energy information and presented by [6], has proven to deliver improved results, by also detecting some of the fine vessel structures, which could be used for further improvements of correctness manually or by a more sophisticated additional approach, like taking into account the geometric structures of vessels and bones.

Porting all presented segmentation methods to parallel algorithms on GPU was also a main task of this thesis and has proven to be quite difficult as well. Although speed gains could be achieved for the simple thresholding algorithms, the speed gains for region growing and the XOR-Algorithm have not turned out to be as expected and partly resulted into speed losses. Skipping one step made it possible to speed up the region growing algorithm and make it faster than its CPU counterpart, despite its dependence on the number of objects detected within the volume. But for the XOR-Algorithm this goal could not be achieved yet, due to its nature of detecting even very small objects in the segmentation process. For future work, to solve this problem, there are several options. Parts of the algorithm could be left on CPU (which would mean additional CPU-GPU transfers). Since the CCL-Algorithm implemented in this thesis is a rather simple one, another solution to the problem could be the implementation of a more sophisticated CCL-Algorithm as presented in [5].

References

[1] NVIDIA Corporation: *NVIDIA CUDA C Programming Guide*: Version 4.0, 2011

[2] Lei Pan, Lixu Gu, Jianrong Xu: *Implementation of Medical Image Segmentation in CUDA*. Shanghai Jiaotong University, Shanghai Renji Hospital, P.R.China, 2008

[3] Armin Markus Kanitsar: *Advanced Visualization Techniques for Vessel Investigation*. Technische Universität Wien, Vienna, 2001-02-13. pp.6-11, 24f, 53-60

[4] Luis Ibanez, Will Schroeder, Lydia Ng, Josh Cates, et al.: *The ITK Software Guide*. Second Edition, Updated for ITK version 2.4, 2005-11-21. pp.503-507, `http://www.itk.org/`, visited on 2012-03-06

[5] Oleksandr Kalentev, Abha Rai, Stefan Kemnitz, Ralf Schneider: *Connected component labeling on a 2D grid using CUDA*. Journal of Parallel and Distributed Computing, 2010-10-14

[6] Alexey Karimov, Andrej Varchola, Gabriel Mistelbauer, Hamed Bouzari: *DECT-based Bone Remover*. Presentation at Institute of Computer Graphics and Algorithms, Technische Universität Wien, Vienna, 2011.

[7] Tom Fawcett: *ROC Graphs: Notes and Practical Considerations for Researchers*. HP Laboratories, CA 94304, 2004

[8] Matus Straka: *Processing and Visualization of Peripheral CT-Angiography Datasets*. Technische Universität Wien, Vienna, 2006-07. pp.42ff

[9] Petr Felkel: *Segmentation of Vessels in Peripheral CTA Datasets: Literature review and first tests of some approaches*. Technische Universität Wien, Vienna.

[10] C. Zahlten: *Reconstruction of branching blood vessels from CT-data*. In B. Urban M. Göbel, H. Müller, editor, Visualization in Scientific Computing, pages 41-52. Springer-Verlag, Wien, 1995.

[11] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever: *Multiscale vessel enhancement filtering*. In W.M. Wells, A. Colchester, and S. Delp, editors, Medical Image Computing and Computer-Assisted Intervention - MICCAI98, volume 1496 of LNCS, pages 130-137. SpringerVerlag, Germany, 1998.

[12] M. Sramek (http://www.viskom.oeaw.ac.at/ milos/): personal communication, Austrian Academy of Sciences