UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI MINISTERUL MUNCII, FAMILIEI ŞI PROTECŢIEI SOCIALE AMPOSDRU
Fondul Social European POSDRU 2007-2013
Instrumente Structurale 2007-2013
MINISTERUL EDUCAŢIEI CERCETĂRII TINERETULUI ŞI SPORTULUI
OIPOSDRU
UNIVERSITATEA "POLITEHNICA" din BUCUREŞTI

FONDUL SOCIAL EUROPEAN
Investeşte în oameni!
Programul Operaţional Sectorial pentru Dezvoltarea Resurselor Umane 2007 – 2013
Proiect POSDRU/88/1.5/S/61178 – *Competitivitate şi performanţă în cercetare prin programe doctorale de calitate (ProDOC)*

UNIVERSITATEA **POLITEHNICA** DIN BUCUREŞTI
Facultatea de Automatică şi Calculatoare
Catedra de Calculatoare

Nr. Decizie Senat ..... din .....

TEZĂ DE DOCTORAT

Analiza și vizualizarea datelor din imagini medicale

Analysis and Visualization of Data from Medical Images

Autor: Ing. Anca-Andreea Morar

COMISIA DE DOCTORAT

Preşedinte	Prof. dr. ing. Adina Florea	de la	Universitatea POLITEHNICA din Bucureşti
Conducător de doctorat	Prof. dr. ing. Florica Moldoveanu	de la	Universitatea POLITEHNICA din Bucureşti
Referent	Prof. dr. ing. Sergiu Nedevschi	de la	Universitatea Tehnică din Cluj
Referent	Prof. dr. ing. Vasile Manta	de la	Universitatea Tehnică „Gh. Asachi” din Iaşi
Referent	Prof. dr. ing. Francisc Iacob	de la	Universitatea POLITEHNICA din Bucureşti

Bucureşti 2012

Contents

ACKNOWLEDGEMENTS

First of all, I want to thank God for the strength, health and wisdom to complete this thesis. "I can do all things through Christ who strengthens me" (Philippians 4:13).

I would like to express my deep and sincere gratitude to my thesis advisor, Prof. Florica Moldoveanu, for her guidance, encouragement and patience during these three years of doctoral study. I wish to give special thanks to Victor Asavei, who significantly contributed to my understanding of graphics programming. I am also grateful to my other colleagues, Alin Moldoveanu, Alexandru Egner and Lucian Petrescu, for their ideas and comments that helped in my research.

Next I would like to thank Prof. Eduard Gröller from The Institute of Computer Graphics and Algorithms at Vienna University of Technology for giving me the opportunity to work in his research group during my doctoral mobility. All the members of the visualization group provided a very nice working environment, and for this I am grateful. A special thank you is extended to Gabriel Mistelbauer for his help and patience during those months in Vienna.

I would also like to express my gratitude to all my family and friends, for the warm encouragements and for the nice moments that helped me regain my energy. I want to acknowledge my debt to Radu Berca, Sebastian Iancu and my husband who read my drafts of the thesis and made instructive comments. My mother Maria and my mother-in-law Rodica took over many of my responsibilities as a mother and a housewife, so that I could dedicate my time to research, and for this I am most indebted to them.

Last, but not least, I want to thank my husband Beni for his love and affection, for all the times he encouraged me and for giving me so many reasons to be grateful. I also want to thank my son Filip for bringing such joy to my life. If not for him, maybe my research would have been better, but because of him, my whole life is better.

INTRODUCTION

Nowadays clinical routine is characterized by a large amount of information, which is especially represented by images. Doctors and radiologists are dealing with both two-dimensional and higher dimensional information like scalar values that are organized in data volumes rather than flat images. Future progress in the clinical workflow that starts from medical image acquisition and ends with diagnosis can be accomplished with the use of computers. Features, as well as other information, can be extracted from medical data with analysis methods. The information can be presented in a perceivable way to the medical experts through visualization. This chapter introduces the motivation of the thesis as well as the challenges of medical image processing, analysis and visualization, providing also an overview of the thesis.

1. MOTIVATION

During the last decades computers have become very useful tools in many domains, of which medicine is one of the most challenging. The process of integration is still in an early stage, as compared to, for example, air transportation or banking. Software applications in medicine can vary from management systems that handle information for patients and doctors to intelligent diagnostic and decision assisting systems, automatic/semi-automatic applications for the analysis of medical tests and health care systems.

Medical image analysis and visualization hold a significant place among medical software applications. Medical data like radiographic and CT images can be processed for diagnosis or treatment prescribing. They can also be used for the visualization of possible results of a surgical procedure like plastic surgery, for assistance during the surgeries or for post-operatory patients follow-up.

Nowadays we can observe an increased interest in the development of automatic medical image analysis applications. Some medical image characteristics like small contrast between different tissues or inhomogeneity within the same tissue increase the difficulty of correctly analyzing them. There are other image imperfections, like those caused by the position shifting of a patient during the scan, but these are not taken into account because they rarely occur.

The human eye can easily distinguish important characteristics in medical images, even in case of very poor quality (blurred and noisy images). The goal of medical software

applications is to interpret medical data in a similar or even better way as compared to a human specialist, but only faster. The results obtained by computers can be very impressive. For example, a CT dataset can be visualized, with the highlighting of some important characteristics like different human tissues. However, these automatic algorithms often lead to errors. Nowadays there are a lot of semi-automatic medical applications that allow domain experts like doctors or radiologists to modify the results produced by computers in case of errors. These applications ease the work of medical staff, but the user interaction is still time consuming. The challenge in this domain is to discover fast and fully automatic medical image processing, analysis and visualization techniques, or algorithms that require very little human interaction.

In this context, our research is focused on developing techniques to handle medical data for the purpose of visualization, 3D reconstruction, segmentation and feature extraction in hip arthroplasty and angiology.

2. MEDICAL IMAGES – BASIC CONCEPTS

According to Lorensen and Cline [1], medical image analysis applications consist of four steps:

1. Data acquisition

This step is performed by the scanning device that samples some properties of a patient and produces a 2D image or a series of images also known as slices. The sampled data depends on the acquisition technique:

- **Radiographic images (X-rays)** allow the study of the whole bone system, joints, abdomen, lungs, etc. Classic radiographic images use X rays. The X-ray device contains a tube that emits X rays, or Röntgen rays, and a plate where the radiographic film is located. Computed radiography or digitized radiography devices use a flexible plate instead of films. This plate is exposed through a laser scanner or reader, which scans and digitizes the image for the transfer to a computer. In the current thesis, computed radiographic images are being referred to as radiographic images or X-rays. The radiation that crosses the objects is attenuated differently depending on the density at a certain point. Thus, the components of the object are discriminated by contrast. For instance, bone tissue has a higher density as compared to other tissues.
- **Ultrasound images** allow the visualization of muscles, tendons, internal organs or fetuses through cyclic sound pressure waves with a frequency greater than the upper limit of human hearing.
- **Computed tomography (CT)** measures the spatially varying X-ray attenuation coefficient, showing internal structure. In conventional CT machines, an X-ray tube and a detector are physically rotated behind a circular shroud. There is another type of CT machine, called electron beam tomography, with a tube that is far larger and has greater power to support high temporal resolution. Unlike the case of radiographic images, body structures appear in a volume or stack of individual images. In 3D medical applications,

computed tomography is frequently used for analyzing bone structure. There are several variations to the conventional CT:

- **Computed tomography angiography (CTA)** is used to visualize blood vessels throughout the body. Usually, a contrast substance is injected into a vein in order to show both blood vessels and other tissues.
- **Dual energy computed tomography (DECT)** uses two X-ray beams, each being set to a different energy level. The concept behind DECT is that structures that produce similar CT attenuation at one beam energy may respond differently to another. From the bone/vessel investigation point of view, low energy CT (or CTA) produces images with a higher contrast between bones/vessels and other tissues, but with a granular aspect. High energy CT produces smoother images, but with a lower contrast between vessels and other tissues.
- **Single-photon emission computed tomography (SPECT)** measures the emission of gamma rays. The source of these rays is represented by a radioisotope distributed within the body. The scan also shows the presence of blood within the structures, with a higher accuracy as compared to conventional CT.
- **Positron emission tomography (PET)** scan is the newest medical image investigation technique. PET images show the metabolism as well as other important body functions, not only the anatomical structure of the organs.

- **Magnetic resonance (MRI)** measures the distribution of hydrogen nuclei and relaxation times of the nuclei. The first physical property shows overall structure within the slices.

The research described in this thesis focuses on the study and improvement of existing image analysis and visualization algorithms, as well as on the implementation of new ones. The techniques were tested on radiographic and CT images. Fig. 1 shows the connection between a radiographic image and a set of CT slices.

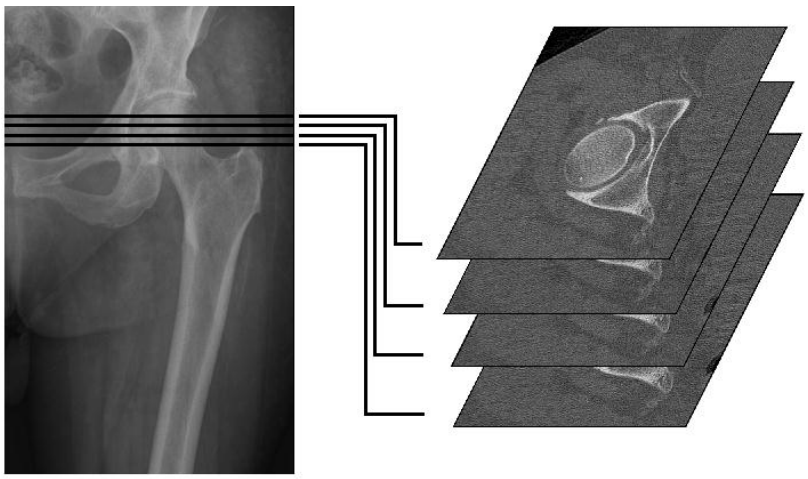

Fig. 1. A vertical view of the human body at the level of the hip provided by a radiographic image (left) and a stack of CT images (right) that represent sections through the scanned body

Images obtained from medical devices have a specific format. In order to manipulate and interpret medical data in an organized manner we chose the most popular medical imaging standard, i.e., DICOM (Digital Imaging and Communication in Medicine) [38]. It contains detailed specifications for coding and transferring medical images and their associated information. Clinical data is represented in various formats: distance is measured in millimeters, time is measured in seconds, etc. Section PS 3.5. of the standard, Data Structures and Their Encoding, defines 27 types of standard data, known as "value representations" (VR). Any information encoded in a DICOM file belongs to one of these predefined types. Some of the most important standard data are: Person Name (PN), Date Time (DT) and Age String (AS). A DICOM file contains information like the name of the patient, the image type, the image size and stores the pixels of the image (or the set of images) obtained from the medical imaging devices.

2. Image processing

Some medical applications use image enhancement techniques for noise removal or contrast improvement. Other image processing applications use segmentation methods in order to highlight certain data structures. Contour detection is another processing technique that can be used for parameter extraction or automatic measurements.

3. Surface reconstruction

Surface reconstruction refers to the process of creating a surface from the 3D data. The model, i.e., the reconstructed surface, consists of volume elements or polygons. This step is accomplished if it is necessary to obtain the 3D model of the scanned components. If we only want to visualize the scanned components, no surface reconstruction is required.

4. Visualization

The medical data is presented to the viewer in a perceivable manner through visualization techniques. The most important are direct volume rendering and indirect volume rendering, which are detailed in the next chapter.

3. MEDICAL FIELDS - HIP ARTHROPLASTY AND VASCULAR DISEASES

As our image processing analysis and visualization applications involve the use of several medical fields and related concepts, it is necessary to start with a short presentation of each of them.

Arthroplasty [2, 3] represents a surgical procedure in which an arthritic or dysfunctional joint surface is replaced with a prosthesis or by remodeling or realigning the joint. The important joint for the research presented in this thesis is the one located at the level of the hip. Fig. 2 presents a femur and a pelvic bone, before and after inserting an artificial implant. The parts of the femoral bone that are of interest in hip arthroplasty are the head, the neck and the body, as illustrated in Fig. 2. Based on the characteristics of these bone parts a prosthesis is inserted in the

interior of the femoral body. The neck and the head of the femur are replaced by the neck and the head of the prosthesis.

The final goal of the research undertaken in this area is to provide an automatic or semi-automatic system that produces the 3D models of customized prostheses based on radiographic and CT images. The 3D models are then prototyped into personalized artificial implants. The characteristics and parameters extracted from medical images in the hip arthroplasty field are analyzed in the next chapters.

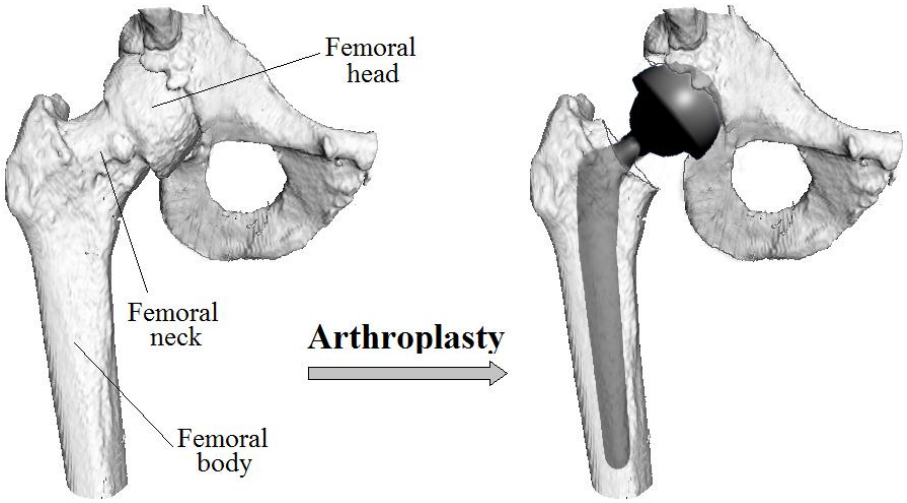

Fig. 2. Hip arthroplasty: before (left) and after (right) inserting a prosthesis

Lower extremity arterial diseases are mainly present in elderly people. Atherosclerotic plaque deposits in the vessel walls narrow the blood flow lumen. Therefore, the blood flow is diminished, resulting in lack of comfort, claudication and in final stage, gangrene.

Kanitsar et al. [35] describe the main arterial diseases that are investigated in the current thesis. Arterial stenoses (Fig. 3(a)) are caused by atherosclerotic plaque and they represent a narrowing of the arterial flow lumen. An occlusion caused by calcification (Fig. 3(b)) denotes a complete obstruction of a vessel. The vessel wall of diseased arteries and the atherosclerotic plaque may present calcifications (Fig 3(c)). In CTA images, calcified tissue is of high attenuation.

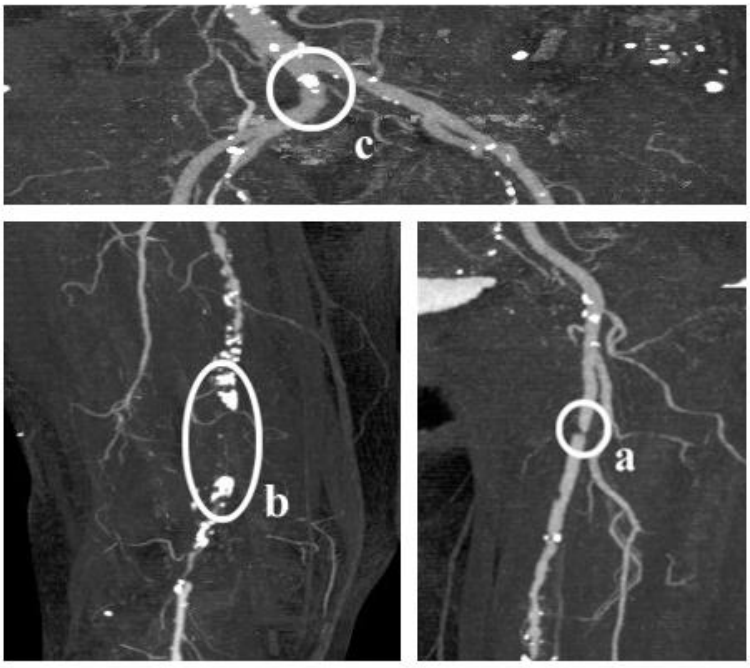

Fig. 3. Arterial diseases: (a) stenosis, (b) occlusion caused by calcification and (c) calcification (source: [35])

4. ORIGINAL CONTRIBUTIONS

This section points out the main contributions of the thesis.

The thesis describes the results of our research in the fields of hip arthroplasty and angiology using image processing and visualization techniques. We studied and improved some existing algorithms and we designed new algorithms and methods of medical data processing and visualization.

We verified our new algorithms and methods using real data from the two medical fields:
- Hip arthroplasty: within the SABIMAS project (http://graphics.cs.pub.ro/SABIMAS/)
- Angiology: within the KASI project (http://www.angiovis.org/)

Our main contributions are:

- a new tool for the investigation of radiographic images that accomplishes automatic and semi-automatic measurements for hip arthroplasty.
- new parallel implementations of the Canny edge detector and the Hough transforms using multi-GPGPU systems and other mechanisms of the CUDA architecture
- the reduction of the search spaces for the Hough transforms based on the particularities of orthopedic radiographic images
- a CUDA real-time volume rendering technique that can handle large datasets

- three non-photorealistic volume rendering techniques that extract and visualize the silhouette from large datasets
- a new 2D image segmentation technique used for discriminating between bones and other tissues, but also between different bones.
- a 3D bone segmentation technique that reconstructs the volume occupied by bones, using CT datasets
- an adaptation of the previously proposed 2D segmentation algorithm for the extraction of bones and vessels from conventional and dual energy CTA
- a method for tracking the vessel tree and extracting the centerlines that are further used for vessel visualization with curved planar reformation.

5. THESIS OUTLINE

The thesis is divided into five chapters.

- The next chapter discusses the state of the art in medical image enhancement, feature extraction, segmentation and 3D visualization, introducing also the GPGPU paradigm.
- Chapter 2 presents the algorithms that we used for automatic and semi-automatic measurements in hip replacement based on radiographic images. The application developed for this purpose extracts parameters that are important in hip arthroplasty based on the particularity that certain parts of bones can be approximated by lines and circles. Parallel implementations of several feature extraction algorithms using the power of multi-GPGPU systems and other particularities of the CUDA architecture are also described.
- Chapter 3 proposes methods for accelerating some volume rendering algorithms with the use of the GPGPU technology. The CUDA implementation of the marching cubes algorithm is customized to handle large datasets by dividing the initial volume into sub-volumes that are processed serially on the GPU. This chapter also describes three silhouette rendering techniques that handle large volumes.
- Chapter 4 introduces a 2D image segmentation method and an algorithm that reconstructs the volume occupied by bones in CT datasets. These image processing techniques are part of a system whose aim is to automatically extract the interior of the femoral bones and generate 3D models of customized prostheses in hip arthroplasty.
- Chapter 5 presents the alterations made to the image segmentation method from chapter 4 in order to handle both conventional and dual-energy CTA datasets for the extraction of bone, as well as vessel tissue. One of the visualization methods for CTA, curved planar reformation, requires the previous tracking of the vessel central axes. This chapter provides an algorithm that segments the bones and the vessels, tracks the vessel tree based on user-defined seed points and extracts the centerlines of the vessels.

CHAPTER 1

STATE OF THE ART

This chapter discusses the state of the art in the research domain of the thesis. It describes methods for image enhancement, feature extraction, image segmentation and visualization of 3D data. It also includes a section that introduces the GPGPU technique which is used in the parallel implementation of image analysis and visualization algorithms.

1.1. MEDICAL IMAGE ENHANCEMENT ALGORITHMS

As previously mentioned, the poor quality of medical images can represent an impediment for their automatic processing. These images usually have a low contrast between different components, and density inhomogeneity within the same component. The noise generated in the image acquisition process also leads to images with a granular aspect. Two noise removing techniques that are further used in the proposed image processing algorithms are detailed in this section.

1.1.1. Noise Removal with Gaussian Filter

Gaussian noise [4] is one of the most common types of noise. It is modeled using a probability density function:

$$PDF(g) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(g-m)^2}{2\sigma^2}\right), \tag{1.1}$$

where g represents the gray level of the current image pixel, m is the noise average and σ is the standard noise deviation. Every pixel in an image with Gaussian noise has a value obtained from summing the real value of the pixel with a random Gaussian density function.

Spatial filtering is used for noise removal as well as for other image processing techniques. This operation is applied locally, at the level of each image pixel, by replacing the value of the current pixel based on the characteristics of the neighboring pixels.

In the proposed image processing algorithms we used convolution masks that approximate the 2D Gaussian distribution function. Fig. 1.1 shows a radiographic image, before and after applying a 5×5 Gaussian convolution mask.

The Gaussian filter removes noise by smoothing the image. However, an unwanted effect of this filter is the smoothing of frontiers in images.

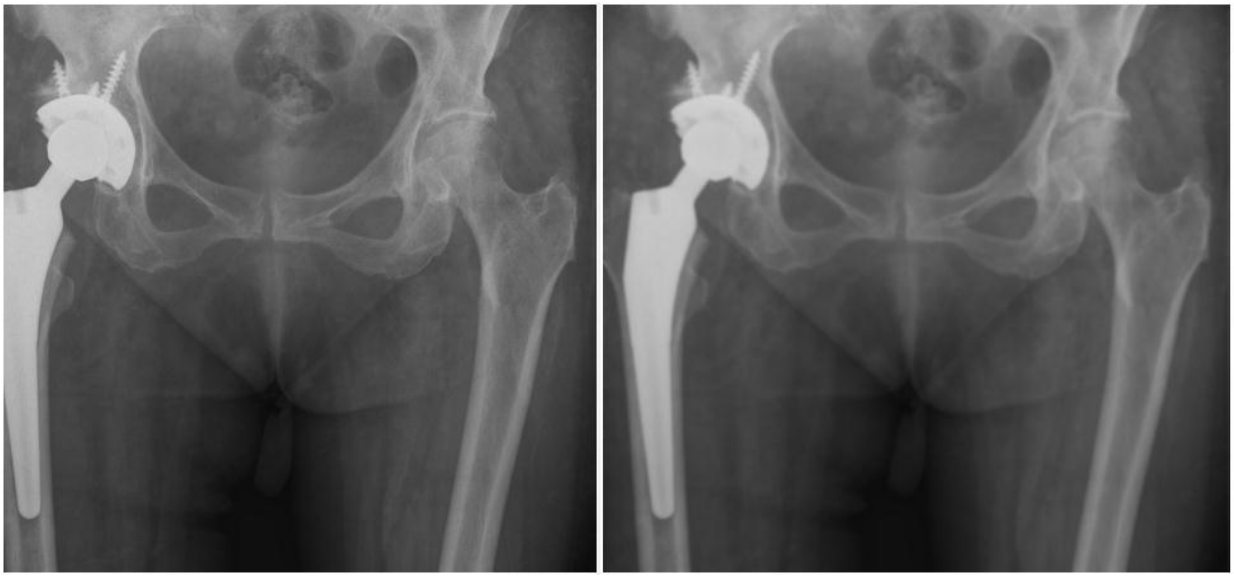

Fig. 1.1. Gaussian filtering: before (left) and after (right) applying a Gaussian filter

1.1.2. Noise Removal Using Nonlinear Anisotropic Diffusion

In contrast to the Gaussian filter, nonlinear anisotropic diffusion [5, 6] reduces noise, but keeps the image information unchanged. Thus, we can smooth images while preserving some characteristics, like edges.

Diffusion is a physical process that balances the concentration changes of a certain substance. Having a concentration distribution u, Flick's law states that the concentration gradient determines a flow whose aim is to compensate the gradient:

$$j = -D \cdot \nabla u \,. \tag{1.2}$$

D is the diffusion tensor, a positive definite symmetric matrix. A real matrix M is positive definite if $z^T \cdot M \cdot z > 0$ for all non-zero vectors z with real entries. z^T is the transpose of z.

Diffusion represents mass transport without destroying or creating new mass. The continuity equation describes the transport of mass:

$$\partial_t u = -div(j) = -\left(\frac{\partial j}{\partial x} + \frac{\partial j}{\partial y}\right), \tag{1.3}$$

where t denotes the time, and $\partial_t u$, the deviation of u with respect to t. The divergence div of a two-dimensional vector field $j = \begin{bmatrix} j_x \\ j_y \end{bmatrix}$ can be computed with the following expression:

$$div(j) = \nabla \cdot j = \frac{\partial j_x}{\partial x} + \frac{\partial j_y}{\partial y}. \tag{1.4}$$

From (1.2) and (1.3) we can deduce the following result:

$$\partial_t u = -div(D \cdot \nabla u). \tag{1.5}$$

In image processing, image intensity can be seen as "concentration" and image noise as concentration inhomogeneities. Depending on the tensor D, diffusion can be linear/nonlinear and isotropic/anisotropic.

1.1.2.1. Linear Isotropic Diffusion

If the tensor D is constant for the whole image, diffusion is linear. If the concentration gradient and the flow are parallel, diffusion is isotropic. The initial image f represents the starting point for the diffusion process:

$$\begin{aligned} &\partial_t u = -div(\nabla u) \\ &u(x, y, 0) = f(x, y). \end{aligned} \tag{1.6}$$

Here, D influences only the speed of the diffusion. Therefore we can choose $D = 1$.

The advantages of linear isotropic diffusion are noise reduction and image simplification. The main drawback of this type of filtering is that it reduces not only noise, but also important image characteristics (frontiers). Fig. 1.2 presents the results of applying linear isotropic diffusion on a CT image.

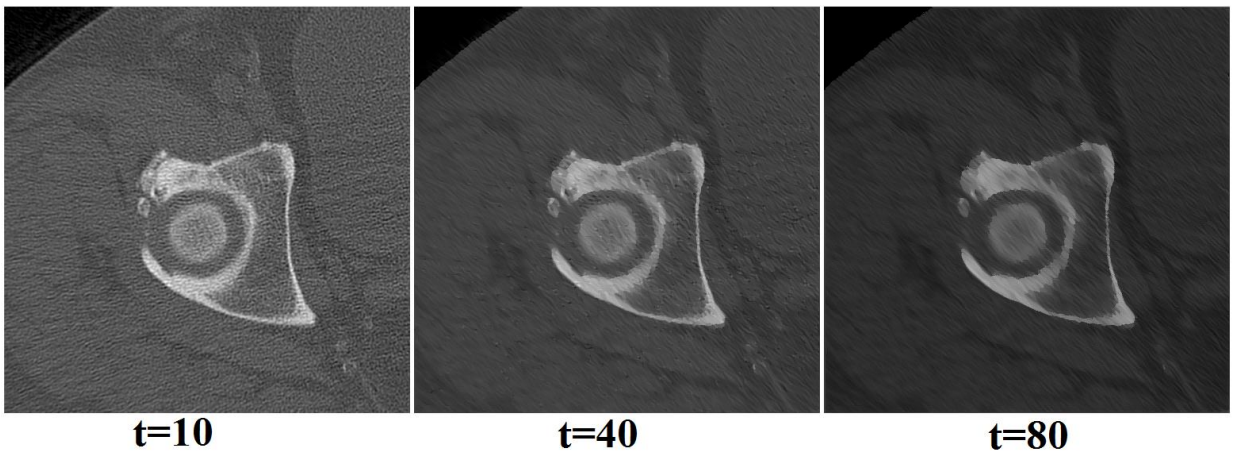

Fig. 1.2. Linear isotropic diffusion filtering of a CT image at different moments in time (different number of iterations)

1.1.2.2. Nonlinear Isotropic Diffusion

As compared to linear isotropic diffusion, nonlinear isotropic diffusion preserves edges to some extent. In this case, the diffusion tensor is a real value (and not a matrix) computed for every pixel.

Depending on the image gradient in the current image, we obtain the following equation:

$$\partial_t u = div(g(|\nabla u|)\nabla u). \tag{1.7}$$

Perona and Malik [7] proposed the following expression for computing the diffusion tensor:

$$D = g(|\nabla u|) = \frac{1}{1+\left(|\nabla u|^2 / \lambda^2\right)}, \tag{1.8}$$

where λ denotes the threshold for the gradient magnitude. Values bigger than the threshold determine edges.

The advantage of this type of filtering is the possibility to define the diffusion parameters for every image pixel. Hence, diffusion can be reduced over edges. However, the smoothing of edges cannot be completely stopped. Fig. 1.3 presents the results of applying nonlinear isotropic diffusion on a CT image.

1.1.2.3. Nonlinear Anisotropic Diffusion

This type of filtering is based on the idea that the tensor D can be defined in such a manner as to avoid certain structures. In case of medical images, diffusion should preserve edges. Therefore, the filtering is accomplished in the following manner:

- no diffusion across edges
- diffusion parallel to the edges

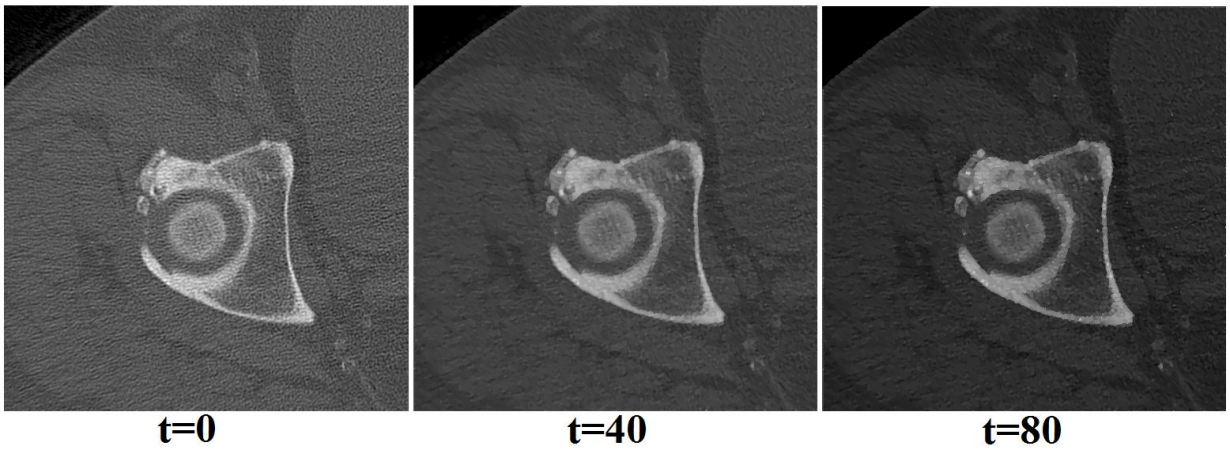

Fig. 1.3. Nonlinear isotropic diffusion filtering of a CT image at different moments in time (different number of iterations)

For a 2D image, the diffusion tensor is a 2×2 matrix determined with the following expression:

$$D = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 \end{bmatrix}^T . \tag{1.9}$$

The eigenvectors v_1 and v_2 provide the local diffusion orientations. Their corresponding eigenvalues give the contrast along these directions. The first eigenvector denotes the direction of the maximum variation. Therefore, v_1 represents the direction parallel to the image gradient. The other eigenvector is determined based on the orthogonality property:

$$v_1 = \frac{\nabla u}{|\nabla u|} \qquad \text{and} \qquad v_2 = \begin{bmatrix} v_{1y} \\ -v_{1x} \end{bmatrix}. \tag{1.10}$$

The eigenvalues are computed so that diffusion is inhibited across the edges, and activated parallel to the edges:

$$\lambda_1 = g\left(\left|\nabla u\right|^2\right) \quad \text{and} \quad \lambda_2 = 1. \tag{1.11}$$

The first eigenvalue is determined using the following expression proposed by Perona and Malik [7] and refined by Weickert [5]:

$$g\left(\left|\nabla u\right|\right) = 1 - \exp\left(-\frac{C_m}{\left(\left|\nabla u\right|^2 / \lambda\right)^2}\right). \tag{1.12}$$

The constant C_m is computed so that diffusion is big for $\left|\nabla u\right|^2 \in [0, \lambda]$ and small for $\left|\nabla u\right|^2 \in (\lambda, \infty)$. λ denotes the threshold for the gradient. Values bigger than the threshold determine edges. In the current implementation of the nonlinear anisotropic diffusion, $\lambda = 4$ and $m = 4$. The value of m determines $C_m = 3.31488$.

After computing the eigenvectors and the eigenvalues of the diffusion tensor $D = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}$, equation (1.5) is solved with finite differences. $\partial_t u$ can be replaced through a forward difference approximation. The obtained explicit scheme allows the iterative computation of further versions of the image:

$$u(x,y,s) = u(x,y,s-1) + \Delta t \cdot \left[\frac{\partial}{\partial x}\left(D_{11}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial x}\left(D_{12}\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial y}\left(D_{21}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_{22}\frac{\partial u}{\partial y}\right)\right]. \tag{1.13}$$

where t denotes the time step size, and $u(x,y,s)$ represents the image at time $t_s = s \cdot \Delta t$. The standard scheme for the approximation of spatial derivatives is based on central differences.

Fig. 1.4 presents the results of applying nonlinear anisotropic diffusion on a CT image. As can be observed, this type of filtering reduces noise, while preserving edges.

Even though nonlinear anisotropic diffusion has many advantages when it comes to medical image enhancement, it does not lead to good results for radiographic images. Fig. 1.5 shows the results of applying nonlinear anisotropic diffusion on an X-ray.

In the example provided in Fig. 1.5, it can be observed that strong edges like those between the prosthesis and other tissue are preserved. On the contrary, smoother and noisy edges like those located at the level of the femoral head, separating the femur from the pelvic bone, are lost. This is the reason we did not use nonlinear anisotropic diffusion for the processing of radiographic images.

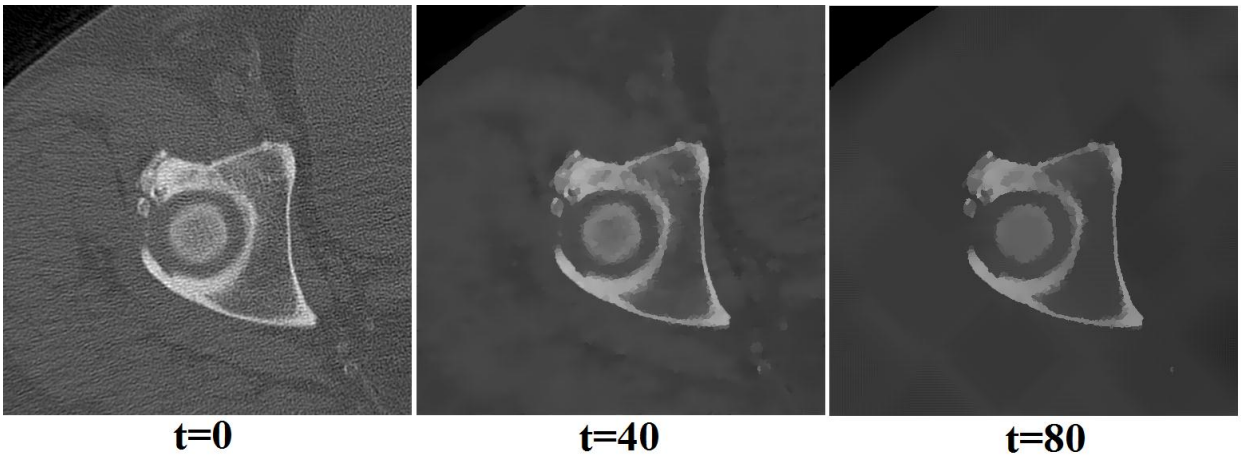

Fig. 1.4. Nonlinear anisotropic diffusion filtering of a CT image at different moments in time (different number of iterations)

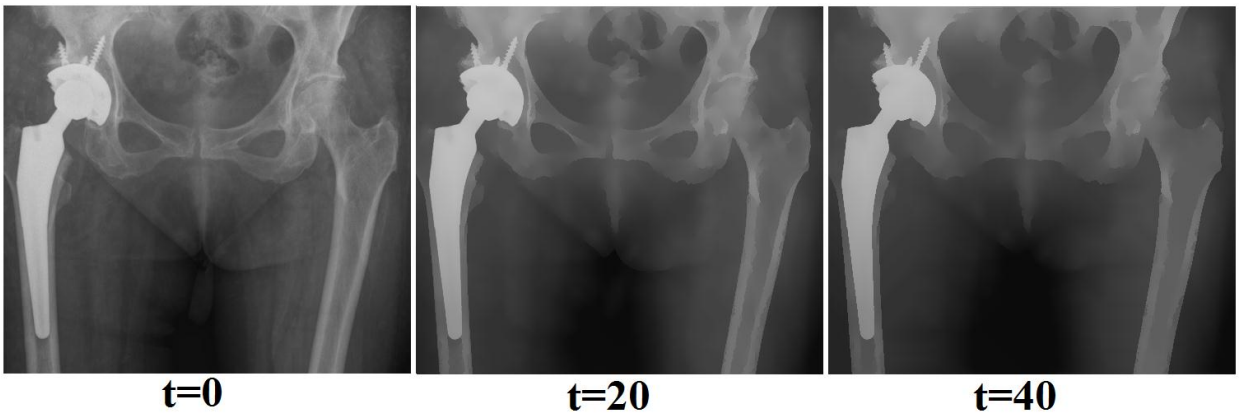

Fig. 1.5. Nonlinear anisotropic diffusion filtering of a radiographic image at different moments in time (different number of iterations)

1.2. FEATURE EXTRACTION FROM IMAGES

A possible approach to automatic feature extraction from images could consist of the following steps:

- detection of candidate edge pixels where the intensity changes abruptly
- removal of candidate pixels that are not edge pixels
- connection of edge pixels in order to obtain contours using
    - local methods that rely on the relationship of each pixel with its neighboring pixels
    - global methods that are based on the knowledge about the shape of the frontiers

- extraction of characteristics.

This section presents the Canny edge detector which extracts contours from images and the Hough transform, a feature extraction technique that finds instances of objects within a certain class of shapes by a voting procedure.

1.2.1. Canny Edge Detection

Pixels where the intensity changes abruptly can be detected by using operators (convolution masks) that approximate the gradient or the Laplacian of the image function $f(x,y)$, which represent the first and second derivative, respectively. We use only the gradient based edge detection.

The gradient vector points in the direction of the largest possible intensity variation at each image pixel. The length of the gradient vector represents the rate of change in that direction. The image gradient is defined based on the partial derivatives of the image function $f(x,y)$:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} D_x \\ D_y \end{bmatrix}. \tag{1.14}$$

The gradient magnitude at pixel (x,y) is defined as follows:

$$D(x,y) = \sqrt{D_x^2 + D_y^2} \tag{1.15}$$

and approximated by:

$$D(x,y) = \max\left(|D_x|, |D_y|\right) \tag{1.16}$$

or by

$$D(x,y) = |D_x| + |D_y|. \tag{1.17}$$

The direction of the gradient is defined by the equation:

$$\theta = \begin{cases} \arctan\left(D_y / D_x\right), & D_y \neq 0, D_x \neq 0 \\ 0, & D_y = 0, D_x \neq 0 \\ \pm 90, & D_y \neq 0, D_x = 0 \end{cases} \tag{1.18}$$

The image gradient can be approximated with finite differences:

$$\begin{cases} D_x = \dfrac{f(x+\Delta x, y) - f(x-\Delta x, y)}{2 \cdot \Delta x} \\ D_y = \dfrac{f(x, y+\Delta y) - f(x, y-\Delta y)}{2 \cdot \Delta y} \end{cases}. \tag{1.19}$$

The most widely used gradient approximation operators are Roberts, Sobel and Prewitt. In the algorithms proposed in this thesis we use both the Sobel operator and central differences (equation (1.19)) for image gradient estimation.

Canny edge detector [8] is a well-known contour extraction method, which produces relatively good results for medical radiographic images. The six steps of the Canny algorithm are briefly described in this section:

1. Noise removal in the original image with Gaussian filtering
2. Gradient computation using the Sobel operator: The gradient magnitude is obtained based on the estimates of the gradient in the horizontal (D_x) and vertical (D_y) direction. The gradient magnitude is computed by using (1.17)
3. Computation of the gradient direction for every pixel based on (1.18)
4. Discretization of the gradient direction with the following expression:

$$\theta = \begin{cases} 0, & if \quad 0 \le \theta \le 22.5 \quad or \quad 157.5 \le \theta \le 180 \\ 45, & if \quad 22.5 \le \theta \le 67.5 \\ 90, & if \quad 67.5 \le \theta \le 112.5 \\ 135, & if \quad 112.5 \le \theta \le 157.5 \end{cases} \tag{1.20}$$

5. Non-maximum suppression by tracing along the edge in the gradient direction and suppressing any pixel that is not a local maximum.
6. Hysteresis thresholding for removal of candidate edge pixels that are noise pixels. It uses two thresholds, T_1 (low) and T_2 (high). Pixels with a value greater than T_2 are considered to be strong edge pixel. Pixels with a value lower than T_1 are considered to belong to the background. Pixels with a value greater than T_2 that are connected to strong edge pixels are also considered to be edge pixels.

Fig. 1.6 shows a radiographic image, before and after applying the Canny edge detector.

Fig. 1.6. Radiographic image before (left) and after applying the Canny edge detector (right)

1.2.2. Hough Transform

The previously described edge detector does not provide information about the shape of the contour. The next step in an image analysis application could be low level feature extraction, which implies approximating frontiers with geometric primitives like lines, circles or ellipses.

We shortly describe the Hough transform method for the detection of lines and circles in binary images, where the contour pixels are set to 1 and the background pixels to 0. The input image for the Hough transform can be obtained through the binarization of the matrix that contains the gradient magnitudes of the initial image. The gradient magnitude matrix can be obtained with convolution masks like Sobel, Roberts or Prewitt. All the pixels with gradient magnitude lower than a given threshold are set to 0 and all the other pixels are set to 1. In our experiments, we have chosen to define the binary image as the output of the Canny edge detector, because it has thinner edges than those obtained with the other operators.

1.2.2.1. Hough Transform for Lines

In image space, a straight line can be described by the equation $y = mx + b$ and can be graphically represented for each pair of points (x, y). The main idea behind the Hough transform is to consider straight lines in terms of parameters, and not as points (x, y) in the image. A straight line $y = mx + b$ can be represented in the parametric space by the pair (b, m). However, there are a series of problems for vertical lines if the parameters b and m are used. This is the reason for choosing another pair of parameters, r and θ. r represents the distance between the line and the image origin, while θ is the angle of the vector from the origin to the closest point

on the line. Fig. 1.7 depicts a straight line defined by the coordinates r and θ. With this parameters, the equation of the line can be written as follows:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right) \qquad \text{or} \qquad r = x\cdot\cos\theta + y\cdot\sin\theta\,. \tag{1.21}$$

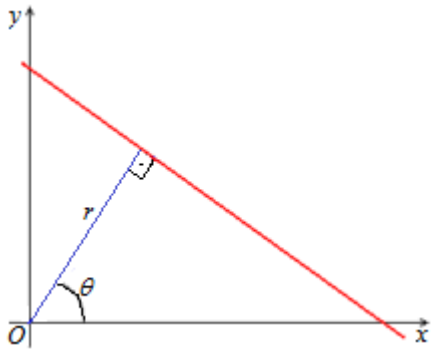

Fig. 1.7. A straight line represented by the parameters *r* and θ

The Hough algorithm uses an array, called accumulator, for the voting procedure. The dimension of the accumulator is equal to the number of unknown parameters. For instance, the Hough transform for lines has two unknown parameters, r and θ. The algorithm determines which pair (r,θ) every pixel belongs to. For every contour pixel (the value of the pixel is 1 in the input image), for every discrete value of θ in the search interval, it looks for a value of r so that the equation (1.21) is satisfied. If there is such an r, the accumulator is incremented at location (r,θ). The pairs (r,θ) where the value of the accumulator is greater than a given threshold determine straight lines.

The complexity of computing the Hough transform for lines is $O(k\cdot N)$, where N is the number of pixels in the input image and k denotes the number of discrete values for parameter θ. For instance, if $\theta = [0,360]$ and the discretization step is 1, then $k = 360$.

The complexity of the algorithm can be reduced with the use of the gradient's direction. If we differentiate equation (1.21) with respect to *x*, we obtain the following expression:

$$\frac{dr}{dx} = \cos(\theta) + \frac{dy}{dx}\cdot\sin(\theta) = 0\,. \tag{1.22}$$

In consequence,

$$\frac{dy}{dx} = -\frac{\cos(\theta)}{\sin(\theta)} = -\cot(\theta) = \tan(\theta + \frac{\pi}{2})\,. \tag{1.23}$$

The gradient's direction for the current pixel (x, y) is $\phi = \arctan(dy/dx)$. So, the parameter θ can be computed based on the direction ϕ:

$$\theta = \phi - \frac{\pi}{2}. \tag{1.24}$$

This method eliminates the "for" loop of parameter θ because for the current pixel (x, y) a single element of the accumulator is updated. Hence, the complexity of the algorithm is reduced to $O(N)$.

1.2.2.2. Hough Transform for Circles

Hough transform for circles is based on the following equation of the circle:

$$(x-a)^2 + (y-b)^2 = R^2, \tag{1.25}$$

where (a,b) is the center of the circle and R is the circle radius. The parameters of the Hough transform for circles are (a,b,R).

The accumulator $A(a,b,R)$ is built based on equation (1.25). For every contour pixel, for every pair (a,b), the algorithm searches for an R so that (1.25) is satisfied. If there is such an R, the accumulator is incremented at location (a,b,R). The triplets (a,b,R) where the value of the accumulator is greater than a given threshold define circles.

The complexity of computing the Hough transform for circles is $O(l \cdot m \cdot N)$, where N is the number of pixels in the input image, l is the number of discrete values of parameter a and m is the number of discrete values of b in the search space.

1.3. MEDICAL IMAGE SEGMENTATION ALGORITHMS

As previously stated, medical images are noisy and have low contrast. These characteristics increase the difficulty of correctly segmenting images. We tried a series of segmentation algorithms for orthopedic radiographic images, in order to segment bones from other tissues and differentiate between bones (femur and pelvis). The low contrast between different tissues and the unclear frontier between bones caused unwanted results. However, in the case of CT images, where there is a better contrast between different tissues, some segmentation algorithms produced relatively good results.

This section discusses the state of the art in image segmentation. It focuses on graph cuts and active contours without edges, two segmentation methods that were compared to our new segmentation algorithm which is detailed in chapter 4.

Boykov and Veksler [9] describe the use of graph cuts in computer vision and graphics through theories and applications. In image segmentation, a graph is created from an image or a set of images. The graph construction and the characteristics that divide the pixels into two disjoint parts, namely the background and the foreground, are detailed in section 1.3.1.

Kass et al. [10] introduce the concept of snakes, or active contours. Snakes are energy-minimizing splines guided by external constraint forces and influenced by image forces that pull them toward features such as lines and edges. Chan and Vese [11] propose active contours without edges. It is a new model for active contours, which is based on techniques of curve evolution, the Mumford-Shah functional [44] for segmentation, and level sets. This method is detailed in section 1.3.2.

In grey scale mathematical morphology, the watershed transform, originally proposed by Digabel and Lantuejoul [12] and later improved by Buecher and Lantuejoul [13], is considered to lead to very good results in image segmentation. Roerdink and Meijster [14] wrote a review of several definitions and algorithms of the watershed transform. They describe the geographic idea behind the watershed transform as that of a landscape or topographic relief which is flooded by water. Watersheds are the dividing lines of the domains of attraction for rain falling over the region. When the water level has reached the highest peak in the landscape, the process is stopped, and the result is a landscape partitioned into basins separated by dams, called watershed lines.

Porwik and Lisowska [15] present the use of the Haar-wavelet transform in digital image processing [45, 46]. Their paper describes a method of image analysis by means of the wavelet-Haar spectrum. Glavasova et al. [16] discuss the wavelet transform for feature extraction, based on texture analysis, for the final goal of image segmentation.

There are a lot of other segmentation algorithms, but most of them are based to some extent on one of the techniques mentioned above. The challenges in image segmentation come from the medical image characteristics previously stated. The small contrast between foreground and background makes it difficult to extract all the edges or lines that are used for example in active contours, graph cuts and watersheds. The inhomogeneities within the objects prove to be a drawback for active contours without edges, because this method tries to minimize the differences within the foreground and the background. The lack of a texture pattern could be a problem for the wavelet transform based segmentation. In chapter 4 we introduce a new algorithm that takes into account the imperfections of poor quality images (like CTs), not only differentiating between objects and background, but also between different objects.

1.3.1. Graph Cuts

The mechanisms of graph cuts for image segmentation are given in more detail. This provides insight into the graph cuts algorithm that was compared to the new segmentation method proposed in chapter 4.

In a tutorial illustrating graph cuts in the context of computer vision and graphics, Boykov and Veksler [9] explain general theoretical properties that motivate their use.

Let $G = \langle V, E \rangle$ be a graph composed of a set of nodes V and a set of oriented edges E. The set $V = \{s,t\} \cup P$ contains two nodes called terminals, i.e., the source s and the sink t, as well as a set of non-terminal nodes P. Fig. 1.8(a) illustrates a graph with its terminals s and t.

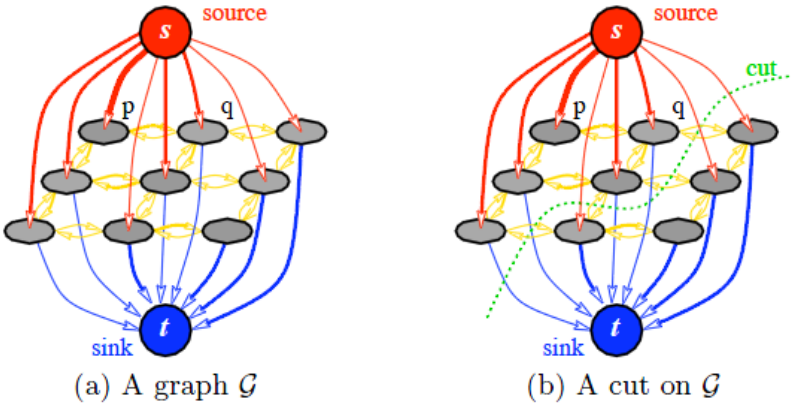

Fig. 1.8. A graph construction based on an image (a); a graph cut (b); source: [9]

Every edge in the graph is assigned a nonnegative weight $w(p,q)$. An edge is called t-link if it connects a non-terminal node with a terminal one. It is called n-link if it connects two non-terminal nodes. An s/t cut is a partitioning of the graph nodes into two disjoint sets S and T in a manner that the source s is in S and the sink t in T. Fig. 1.8(b) illustrates a cut for the graph in Fig. 1.8(a). The cost of a cut $C = \{S,T\}$ is the sum of weights for the frontier edges (p,q), so that $p \in S$ and $q \in T$. The minimal cut problem is to find a cut with the minimum cost. The minimal cut problem can be solved by finding the maximum flow from the source to the sink. The value of the maximum flow is equal to the cost of the minimal cut.

Starting from an image, the graph is built in a manner that every pixel in the image defines a non-terminal node of the graph. The cost of an n-link is based on the "likeliness" of the neighboring non-terminal nodes. The cost of a t-link is based on the "likeliness" of the connected non-terminal node and the terminal one. After the cut, some pixels belong to the source (being labeled as "object") and others to the sink ("background"). The minimal cut minimizes the energy function introduced by Boykov and Veksler [9]:

$$E(f) = \sum_{(p,q) \in Q} V_{p,q}(f_p, f_q) + \sum_{p \in P} D_p \cdot f \ , \tag{1.26}$$

where D_p is the per-pixel term that reflects the penalty of assigning the label f_p to pixel p. $V_{p,q}$ is the border term that encourages spatial coherence within the objects and the background. Q represents the set of all n-links, f_p is the label assigned to pixel p ("object" or "background") and P denotes the set of all non-terminal nodes.

For bone segmentation in CT images, we define a series of rules for computing the per-pixel term and the border term based on the intensities of the pixels and the image gradient.

In order to define the border term $V_{p,q}$ we use the observation of Boykov and Veksler [9] that pixels with high image gradient would imply low cost of n-links and vice-versa. This is why the gradient of the image is computed with a Sobel filter. For smoother borders, the Sobel filter is applied after convolving the image with a Gaussian filter and a nonlinear anisotropic diffusion filter. If the absolute difference between the gradient magnitude of two neighboring pixels p and q is greater than a given threshold k, $V_{p,q}$ is directly proportional with that difference. If there is a small variation of the gradient, the value of $V_{p,q}$, given by the constant v, is high. The border term is defined as follows:

$$V_{p,q} = \begin{cases} \dfrac{1}{\left\|\nabla u_f(p)\right| - \left|\nabla u_f(q)\right\|}, if \left\|\nabla u_f(p)\right| - \left|\nabla u_f(q)\right\| > k \\ v, otherwise \end{cases}, \tag{1.27}$$

where u_f is the image obtained after applying the Gaussian filter and the nonlinear anisotropic diffusion filter. This definition encourages borders in regions where there are abrupt variations of the gradient magnitude.

Boykov and Jolly [17] designed a technique for general purpose interactive segmentation of N-dimensional images. In their method, the user marks certain pixels as "object" or "background". We extend their approach by marking certain pixels based on their intensities. We introduce two thresholds, T_{low} and T_{high}, that determine the most probable background and foreground pixels. If the intensity of a pixel is greater than T_{high}, the pixel is most likely an object pixel. If its intensity is lower than T_{low}, there is a high probability that the pixel belongs to the background. If the pixel does not belong to any of the two categories, the per-pixel term depends on its intensity relative to T_{low} and T_{high}:

$$D_p(f_p) = \begin{cases} w, if \ \ f_p = "object" \ and \ u_0(p) > T_{high} \\ 0, if \ \ f_p = "background" \ and \ u_0(p) > T_{high} \\ w, if \ \ f_p = "background" \ and \ u_0(p) < T_{low} \\ 0, if \ \ f_p = "object" \ and \ u_0(p) < T_{low} \\ w \cdot \dfrac{u(p) - T_{low}}{T_{high} - T_{low}}, if \ \ f_p = "object" \\ \qquad\qquad and \ T_{low} \le u_0(p) \le T_{high} \\ w - w \cdot \dfrac{u(p) - T_{low}}{T_{high} - T_{low}}, if \ \ f_p = "background" \\ \qquad\qquad and \ T_{low} \le u_0(p) \le T_{high} \end{cases}, \tag{1.28}$$

where u_0 denotes the initial image. The constant w assures a high cost for t-links that connect a non-terminal node and a terminal one that have the same label (either "object" or "background").

We implemented two segmentation algorithms based on graph cuts. In the first implementation, the set of n-links Q contains only 4-connected neighbors. The second implementation can be used only for volumetric data, because it also considers connections of pixels from adjacent slices. The first method is further called 2D graph cuts, because it segments the images individually. The second one is being referred to as 3D graph cuts, because it can be applied only to 3D images, like CT datasets.

Fig. 1.9 shows the results of applying the 2D and 3D graph cuts segmentation algorithms on a series of consecutive CT slices. Both methods have difficulties in differentiating between bones (femur and pelvis) and between bones and other tissues. These problems are caused by the small contrast between bones and other tissues and the small distance between bones, especially at the area of the hip joint. Another cause is the inhomogeneity within the bones. As can be observed in the first row of images from Fig. 1.9, there is a big difference in intensity between cortical bone tissue (high intensity) and trabecular bone tissue (low intensity).

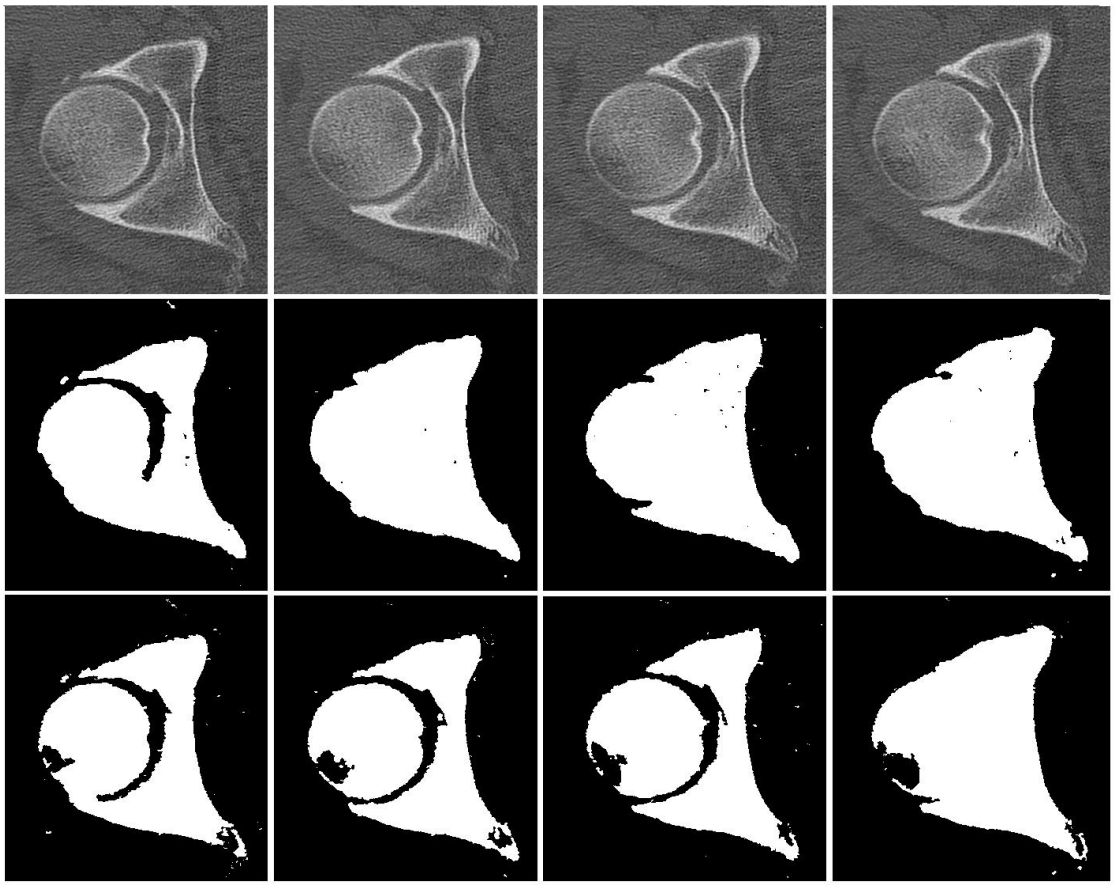

Fig. 1.9. Image segmentation with graph cuts on consecutive CT images: initial images (first row), images obtained with 2D graph cuts (second row) and images obtained with 3D graph cuts (third row)

1.3.2. Active Contours without Edges

This section presents the active contours model without edges. Part of the method is included in the flow of the proposed segmentation algorithm from chapter 4. This technique is also used as a comparison to our segmentation. In their paper, Chan and Vese [11] propose a new active contour model for object detection in a 2D image. The stopping term for the curve evolution process does not depend on the image gradient, but is related to a particular segmentation of the image.

An evolving curve C in the image space Ω can be defined as the frontier of a subset ω of Ω ($\omega \subseteq \Omega$ and $C = \partial\omega$). ω represents the region occupied by foreground pixels. $inside(C)$ denotes the region ω and $outside(C)$ denotes the region $\Omega \setminus \overline{\omega}$. The image u_0 is assumed to be composed of two regions of approximately constant intensities c_1 (the intensity of the object) and c_2 (the intensity of the background). If the object's boundary is C, then inside of C the intensity value should be equal to c_1. Outside of C, the intensity value should be equal to c_2. Chan and Vese [11] introduce the following energy:

$$F(c_1, c_2, C) = \mu \cdot Length(C) + \nu \cdot Area(inside(C)) + \eta_1 \int_{inside(C)} |u_0(x,y) - c_1|^2 dxdy + \eta_2 \int_{outside(C)} |u_0(x,y) - c_2|^2 dxdy, \quad (1.29)$$

where $\mu \geq 0$, $\nu \geq 0$, $\eta_1, \eta_2 > 0$ are fixed parameters. The length of the curve, $Length(C)$, and the area of the region inside C, $Area(inside(C))$, are two regularizing terms. Chan and Vese [11] set $\nu = 0$, $\eta_1 = 1$ and $\eta_2 = 1$.

The image segmentation into foreground and background is accomplished by solving the minimization problem $\inf_{c_1, c_2, C} F(c_1, c_2, C)$.

Let $C \subset \Omega$ be defined as the zero level set of a Lipschitz function $\phi : \Omega \to R$, so that:

$$\begin{cases} C = \partial\omega = \{(x,y) \in \Omega : \phi(x.y) = 0\} \\ inside(C) = \omega = \{(x,y) \in \Omega : \phi(x.y) > 0\} \\ outside(C) = \Omega \setminus \overline{\omega} = \{(x,y) \in \Omega : \phi(x.y) < 0\} \end{cases}. \quad (1.30)$$

Using the Heaviside function $H = \begin{cases} 1, if\ z \geq 0 \\ 0, if\ z < 0 \end{cases}$, and the Dirac measure $\delta_0 = \frac{\partial}{\partial z} H(z)$, the energy $F(c_1, c_2, C) = F(c_1, c_2, \phi)$ can be expressed as follows:

$$F(c_1,c_2,C)=\mu\int_{\Omega}\delta_0(\phi(x,y))\nabla|\phi(x,y)|dxdy+\nu\int_{\Omega}H(\phi(x,y))dxdy+$$
$$\eta_1\int_{\Omega}|u_0(x,y)-c_1|^2H(\phi(x,y))dxdy+\eta_2\int_{\Omega}|u_0(x,y)-c_2|^2(1-H(\phi(x,y)))dxdy \quad . \tag{1.31}$$

The constants c_1 and c_2 can be expressed relative to ϕ:

$$c_1(\phi)=\frac{\int_{\Omega}u_0H(\phi(x,y))dxdy}{\int_{\Omega}H(\phi(x,y))dxdy} \quad \text{and} \tag{1.32}$$

$$c_2(\phi)=\frac{\int_{\Omega}u_0(1-H(\phi(x,y)))dxdy}{\int_{\Omega}(1-H(\phi(x,y)))dxdy}. \tag{1.33}$$

The evolution of ϕ can be parameterized based on the following equation:

$$\begin{cases}\dfrac{\partial\phi}{\partial t}=\delta_0(\phi)\left[\mu\cdot div\left(\dfrac{\nabla\phi}{|\nabla\phi|}\right)-\nu-\eta_1(u_0-c_1)^2+\eta_2(u_0-c_2)^2\right]=0\ in\ (0,\infty)\times\Omega\\ \phi(0,x,y)=\phi_0(x,y)\ in\ \Omega\\ \dfrac{\delta_0(\phi)}{|\nabla\phi|}\cdot\dfrac{\partial\phi}{\partial\vec{n}}=0\ on\ \partial\Omega\end{cases} \tag{1.34}$$

where $\vec{n}$ denotes the exterior normal to the boundary $\partial\Omega$, and $\partial\phi/\partial\vec{n}$ is the normal derivative of at the boundary.

The Heaviside function and the Dirac measure are approximated by the following expressions:

$$H_\varepsilon(z)=\frac{1}{2}\left(1+\frac{2}{\pi}\arctan\left(\frac{z}{\varepsilon}\right)\right)\text{and}\quad \delta_\varepsilon=H_\varepsilon',\ \text{where}\quad \varepsilon=0.001. \tag{1.35}$$

The segmentation algorithm follows an iterative method. Knowing $\phi(x,y,s)$ at time $t_s=s\cdot\Delta t$, $c_1(\phi,s)$ and $c_2(\phi,s)$ can be determined by using (1.32) and (1.33). Then, $\phi(x,y,s+1)$ are computed with finite differences. More details on the computation of ϕ are given in [11].

Fig. 1.10 shows the result of segmenting a series of consecutive CT slices with active contours method without edges.

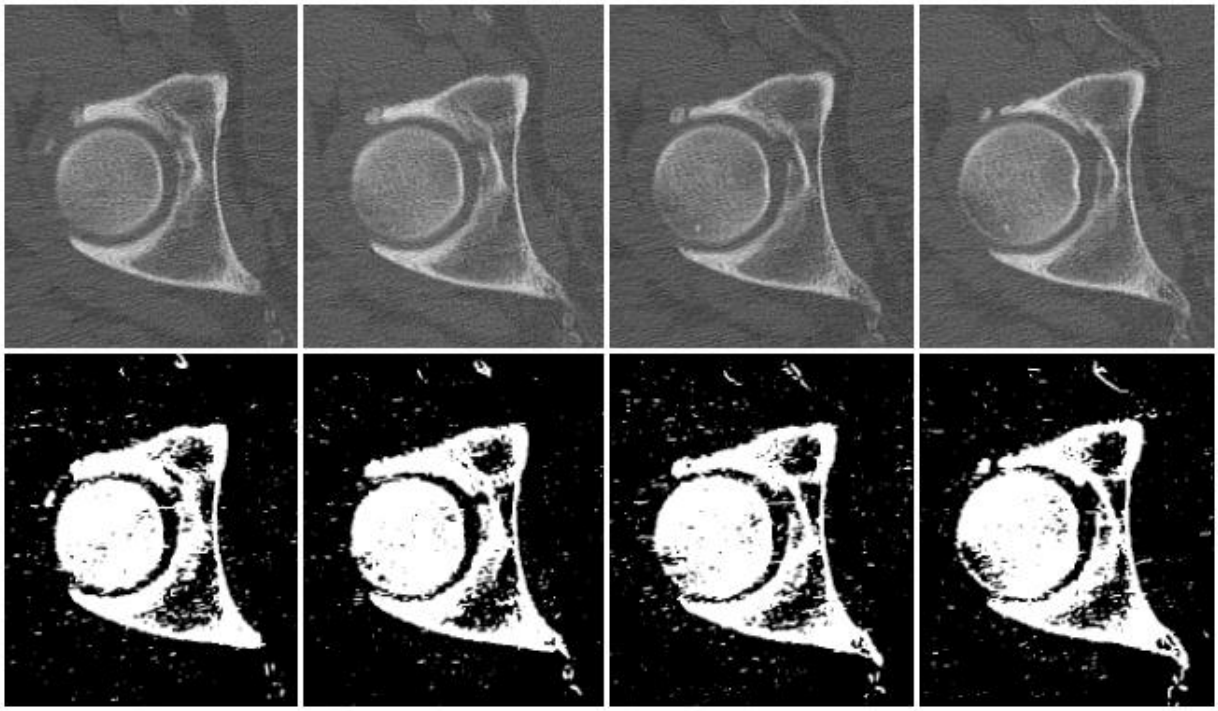

Fig. 1.10. Image segmentation with active contours without edges on a series of consecutive CT images: initial images (first row) and images obtained after applying the segmentation algorithm (second row)

As can be observed, bone segmentation in CT images using active contours without edges produces better results than the segmentation with graph cuts. However, the problem of discriminating between different bones is still present.

We applied a morphological opening (erosion followed by dilation) for the removal of the pixels that wrongly connect different bones. Most of the noise pixels and the pixels connecting the femoral bone and the pelvis were removed. However, the morphological opening caused some discontinuities in the object contour. Fig. 1.11 presents an image segmented using active contours without edges, before and after applying the morphological operation. It can be observed that the tradeoff between removal of wrongly labeled foreground pixels and discontinuities that appear in the object is not desirable.

The problems of the existing segmentation algorithms led to the idea of designing a new algorithm that takes into account the characteristics of CT images, not only differentiating between bones and other tissues, but also between different objects. The new segmentation algorithm is detailed in chapter 4.

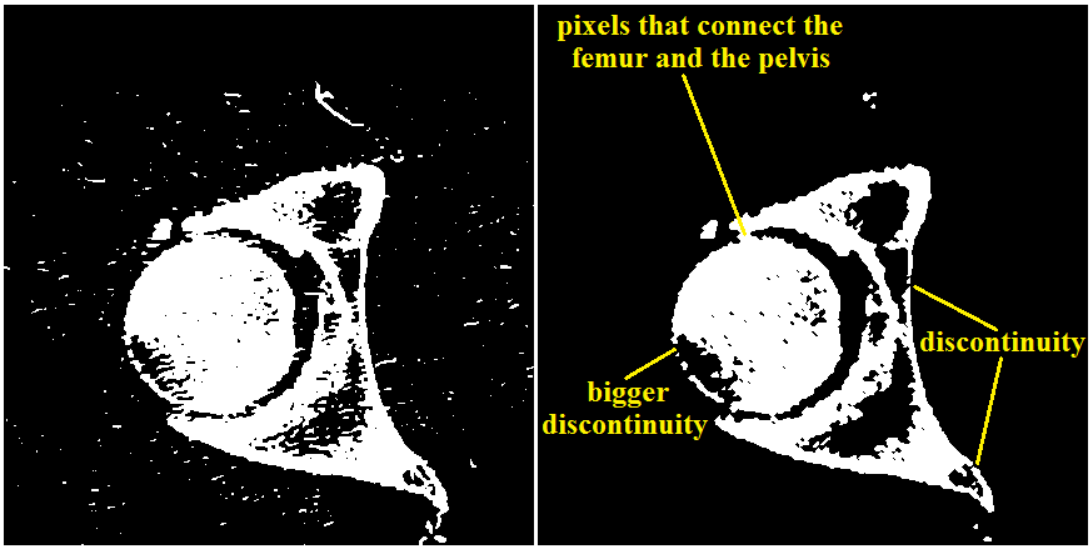

Fig. 1.11. CT image segmented with the active contours algorithm without edges, before (left) and after applying a morphological opening (right)

1.4. VOLUME RENDERING

Most 3D applications allow the visualization of the data volume that consists of a stack of images. In medicine, volume rendering is used for the inspection of internal structures of the scanned body. Visualization can be helpful in diagnosis or in showing the possible result of a surgery. A data volume is obtained from a medical acquisition device, in form of a stack of 2D images that represent sections through the scanned body. The dataset is geometrically represented by a volume composed of voxels. Every voxel vertex has a scalar value obtained from the dataset. Fig. 1.12 illustrates the geometric representation of the dataset.

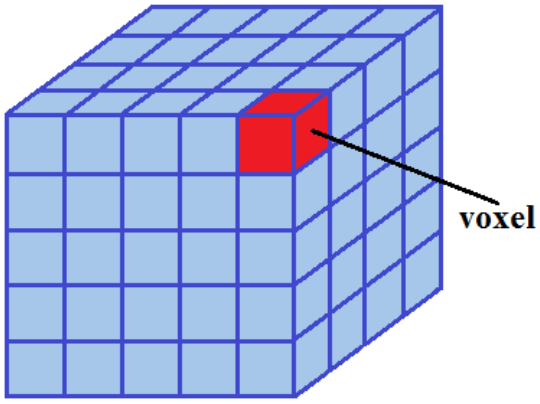

Fig. 1.12. Geometric representation of the data volume

This section describes some of the most frequently used volume rendering techniques, ray casting and marching cubes, and discusses the state of the art in non-photorealistic volume rendering.

1.4.1. Direct Volume Rendering – Ray Casting

The volume ray casting method, described by Levoy [18], casts rays from the eye to each image pixel into the volume, sampling it at certain intervals. The color and opacity are computed for every sample, based on a transfer function. The color and opacity are then composited along each ray.

Amongst the advantages of this method we can mention the high quality of the produced image, with a low memory usage. This algorithm is easy to implement and very fast. One of the main drawbacks is that it can be used only for visualization. If, for instance, we want to obtain the 3D model of the scanned body, we need different rendering methods.

1.4.2. Indirect Volume Rendering – Marching Cubes

Marching cubes [1] is a surface reconstruction algorithm. It extracts an iso-surface from the volume of voxels, computing its intersection with the frontier voxels. An iso-surface is a surface composed of points in the volume that have the same scalar value which is also known as iso-value. The vertices of a voxel can be:

- Exterior to the surface (the scalar value of the vertex is smaller than the iso-value)
- On the iso-surface (the scalar value of the vertex is equal to the iso-value)
- Inside the iso-surface (the scalar value of the vertex is greater than the iso-value)

A frontier voxel contains both exterior and interior vertices. Fig. 1.13 depicts the intersection of a voxel with the iso-surface, based on the values of the voxel's vertices.

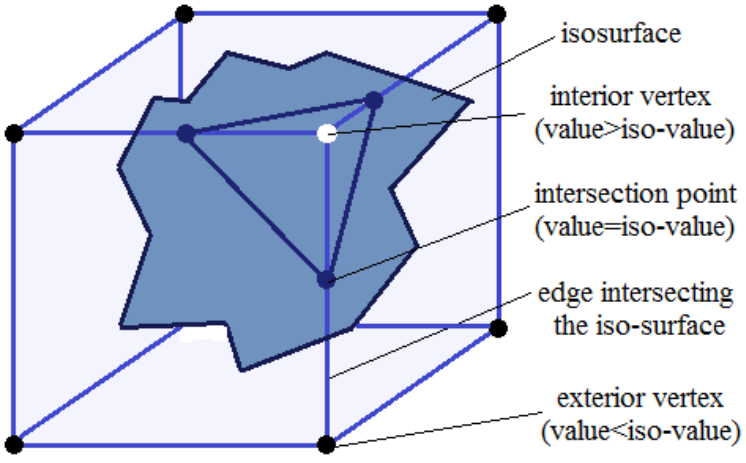

Fig. 1.13. The intersection of an iso-surface with a volume voxel

There are 256 possible configurations for the vertices of a voxel relative to the iso-surface. The triangular surface that approximates the iso-surface can be defined for each configuration. Two search tables are used for coding the 256 configurations:

- the edge table, which defines the edges that intersect the iso-surface
- the triangle table, which defines the way the intersection points are connected

The coordinates of the intersection points are computed using linear interpolation between the extremities of the intersected edges.

Marching tetrahedrons [47] is a variation of the marching cubes algorithm. It divides each cube in five irregular tetrahedrons. The method clarifies a minor ambiguity of the marching cubes algorithm for certain cube configurations. However, due to the cube division into five regions, the number of generated triangles is bigger than the number of triangles obtained with the classic marching cubes approach. In consequence, the iso-surface rendering time increases.

The main advantage of the indirect volume rendering methods is the generation of the geometric 3D model of the scanned body. The reconstructed surface can be used in the medical field for artificial implant prototyping or for visualizing the result of certain surgeries. Among the disadvantages we mention the complexity of the algorithm, which implies a high memory usage and large computing times for the surface generation.

Fig. 1.14 depicts the reconstruction of a volume obtained from a CT dataset.

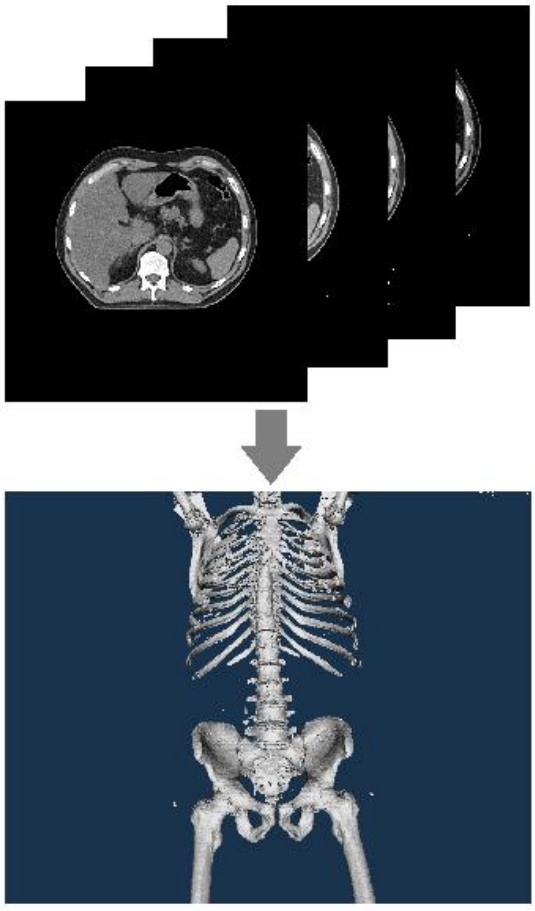

Fig. 1.14. The reconstruction of a volume with marching cubes

1.4.3. Non-photorealistic Volume Rendering

This section presents a series of non-photorealistic volume rendering methods that can be used in medicine in order to highlight important characteristics in a volume dataset.

Some non-photorealistic volume rendering techniques use iso-surface reconstruction. Pelt et al. [19] propose an interactive GPU illustrative framework based on a particle system that is created by using a marching cubes like approach.

Dong et al. [20] introduce volumetric hatching, a technique that produces pen-and-ink style images from medical 3D images by generating iso-surfaces from the data with marching cubes and applying standard surface-hatching techniques.

Yuan and Chen [21] illustrate surfaces by drawing feature lines, such as silhouettes, valleys, ridges and surface hatching strokes. This approach first extracts a point-based surface model described by Co et al. [22] and then renders the volume, embedding surfaces from the first step.

Other methods, based on the ray casting algorithm, modify the transfer function in order to obtain non-photorealistic renderings. Luo et al. [23] present an illustrative technique that focuses on a user-driven region of interest with a distance function that controls the opacity of the voxels.

Kindlmann et al. [24] advance the use of curvature information in multi-dimensional transfer functions, bringing contributions to three different application areas: non-photorealistic volume rendering, surface smoothing via anisotropic diffusion and visualization of iso-surface uncertainty.

Bruckner and Gröller [25] propose the concept of style transfer functions that enable flexible data-driven illumination. This method goes beyond the use of the transfer function only for assigning colors and opacities.

Hadwiger et al. [26] describe a real-time rendering pipeline for implicit surfaces defined by a volumetric grid using a ray-casting approach and local shape descriptors.

Burns et al. [27] render silhouettes from volume data with a voxel-based marching lines technique that traces contours. We implemented three silhouette extraction algorithms from volumes, which are described in chapter 3, based on this approach. In their paper, Burns et al. propose a CPU method for the extraction of silhouettes and suggestive contours. They determine the silhouette by intersecting the iso-surface with the surface where the normal is perpendicular to the view ($\overline{N} \cdot \overline{V} = 0$). Fig. 1.15 illustrates the extraction of the silhouette for a volume voxel with their algorithm.

Vertices A_0, A_2 and A_3 are outside the iso-surface because their scalar values $a_0, a_2, a_2 < iso-value$, and A_1 is inside the iso-surface because $a_1 > iso-value$. Based on the scalar values of the voxel's vertices, a triangle is generated, with the vertices B_0, B_1 and B_2 and their normal vectors $\overline{N}_0$, $\overline{N}_1$ and $\overline{N}_2$.

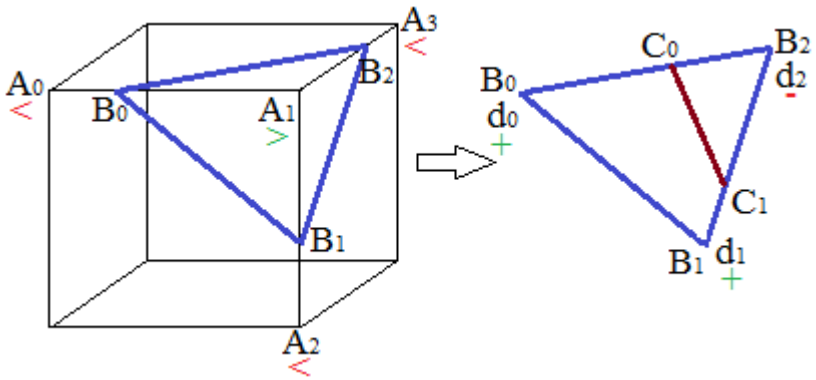

Fig. 1.15. Extraction of a silhouette segment from a volume voxel

Let $\overline{V} = eye - B$ be the view vector for vertex B. The dot product $d = \overline{N} \cdot \overline{V}$ is then computed for every triangle vertex. If the dot products have different signs (for example, d_0 and d_1 are positive, d_2 is negative), this means that there is a silhouette segment $[C_0C_1]$ inside the triangle. The silhouette segment is determined using linear interpolation from the triangle vertices, based on d_0, d_1 and d_2. If vertex B_0 has the coordinates (x_{b0}, y_{b0}, z_{b0}) and vertex B_2 has the coordinates (x_{b2}, y_{b2}, z_{b2}), then the coordinates (x_{c0}, y_{c0}, z_{c0}) of silhouette vertex C_0 are computed as follows:

$$\begin{cases} x_{c0} = x_{b0} + (x_{b2} - x_{b0}) \cdot \dfrac{0 - d_0}{d_2 - d_0} \\ y_{c0} = y_{b0} + (y_{b2} - y_{b0}) \cdot \dfrac{0 - d_0}{d_2 - d_0} \\ z_{c0} = z_{b0} + (z_{b2} - z_{b0}) \cdot \dfrac{0 - d_0}{d_2 - d_0} \end{cases}. \tag{1.36}$$

After obtaining the lines from the volume, a visibility test is realized by casting rays from the endpoints of each silhouette segment to the eye. For each volume voxel intersected by a ray, the algorithm determines whether it intersects the iso-surface in that cell by examining the intersection with the face by which it leaves the cell. At that intersection, the function value is computed with linear interpolation from the four corners of the face in order to determine whether it has changed sign, indicating that the ray has passed from outside to inside the iso-surface. If it has, the originating vector is marked as occluded.

1.5. GPGPU PARALLELISM

This section discusses base concepts of parallel programming, focusing on the GPGPU paradigm and the CUDA architecture.

During the last few years, the programmable GPU (Graphic Processor Unit) has evolved into a highly parallel processor with great computational power, as illustrated in Fig. 1.16 [28].

The discrepancy in floating-point capability between the CPU and the GPU is caused by the fact that the GPU is specialized for highly intensive parallel computation. Thus, more transistors are devoted to data processing rather than data caching and flow control.

Parallel computations refer to the execution of the same program on many data elements in parallel with high arithmetic intensity. This means that the arithmetic operations are more frequent than the memory access operations. As most of the medical image analysis applications process all the image pixels/volume voxels in the same manner, the GPGPU (General Purpose Graphic Processor Unit) paradigm can be very useful in this field.

CUDA (Compute Unified Device Architecture) [28] is a parallel computing architecture that leverages the parallel compute engine in Nvidia GPUs to solve many complex computational problems in a more efficient way than on the CPU.

The challenge in the GPGPU field is to develop software applications that transparently scale their parallelism in order to use many-core GPUs with varying number of cores. The CUDA parallel programming model is designed to meet this challenge and to be easily used by programmers familiar with standard programming languages like C.

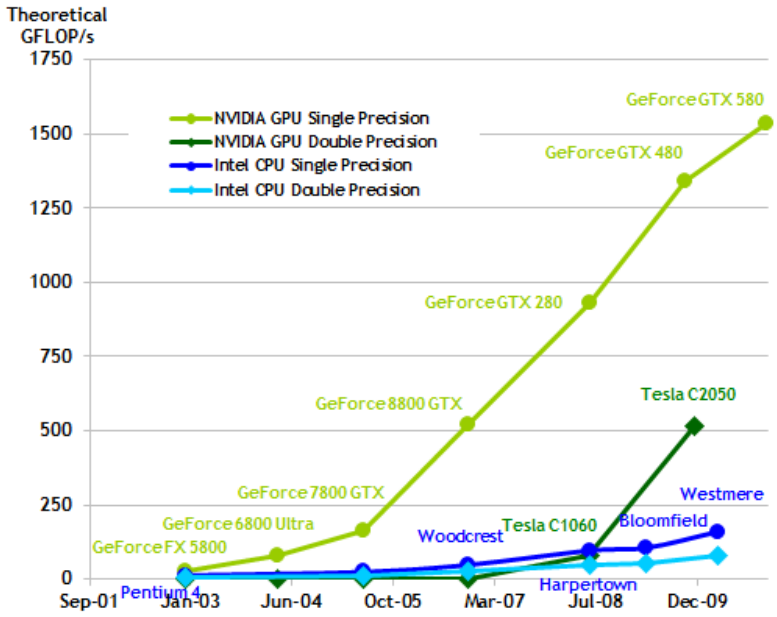

Fig. 1.16. Floating-point operations per second (source: [28])

At the core of CUDA there are three main abstractions that are exposed to the programmers as a minimal set of language extensions. These abstractions are a hierarchy of thread groups, shared memory and barrier synchronization. The abstractions allow the programmer to partition the problem into sub-problems that can be solved cooperatively in parallel by all the threads of a group. The partitioning allows threads to cooperate when solving each sub-problem, and also enables automatic scalability. Each thread block can be assigned to any of the available GPU cores, in any order, concurrently or sequentially, so the CUDA program can be executed on any number of multiprocessors, as illustrated in Fig. 1.17.

Further we describe some main concepts behind the CUDA programming model by outlining the way they are exposed in C.

1. Kernels

CUDA C extends the well-known C programming language by allowing the definition of functions, called kernels, that, when called, are executed in parallel by different CUDA threads, unlike the regular C functions that are executed only once.

A kernel is defined by the specifier `_global_`. The number of CUDA threads that execute the kernel is set by using a `<<<...>>>` syntax. Each thread that executes the kernel has a unique identifier that is accessible within the kernel through the built-in variable `threadIdx`.

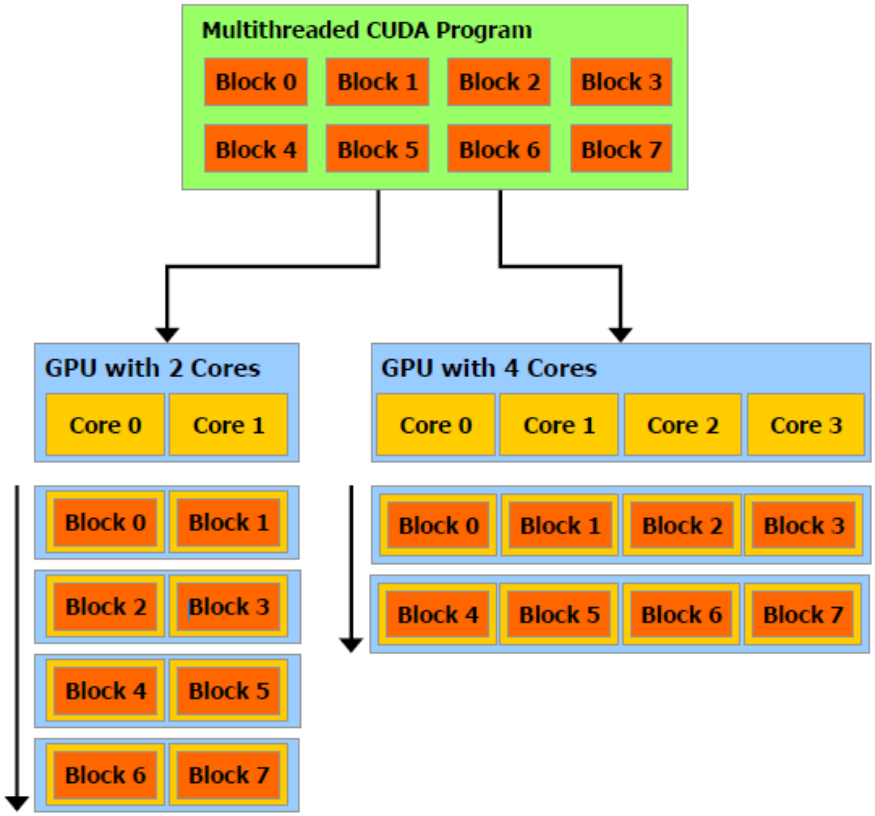

Fig. 1.17. Automatic scalability: A multithreaded program is decomposed into blocks of threads that are executed independently from each other, so a GPU with more multiprocessors will execute the program in less time than a GPU with less multiprocessors (source: [28])

2. Thread hierarchy

`threadIdx` is a vector with 3 components, so the threads can be identified with a one-, two- or three-dimensional index. The block of threads can be one-, two- or three-dimensional. This provides a natural way to invoke computation across the elements in a domain such as a vector, a matrix or a volume.

A thread index and it's ID relate to each other in the following way:

- for a one-dimensional block, the index and the thread ID are the same
- for a two-dimensional block of size (D_x, D_y), the ID of a thread with the index (x, y) is $(x + y \cdot D_x)$
- for a three-dimensional block of size (D_x, D_y, D_z), the ID of the thread with the index (x, y, z) is $(x + y \cdot D_x + z \cdot D_x \cdot D_y)$.

Blocks are organized into one-, two- or three-dimensional grids. The number of blocks in a grid is usually based on the size of the processed data rather than the number of GPU multiprocessors. Fig. 1.18 depicts the structure of two-dimensional grids that contain two-dimensional blocks of threads.

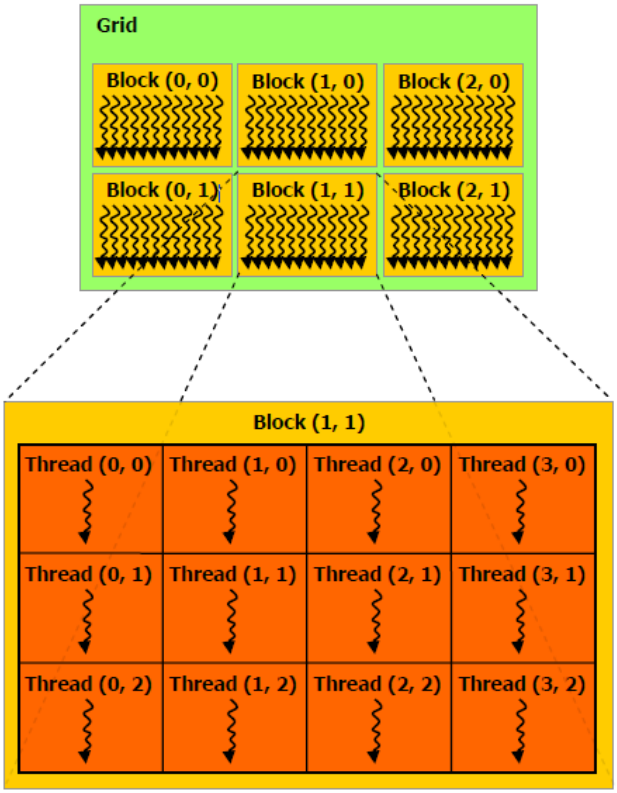

Fig. 1.18. Two-dimensional grid of two-dimensional blocks of threads (source: [28])

The number of threads in a block and the number of blocks in a grid are set with the `<<<...>>>` syntax and can be of type `int` or `dim3`. Each block in a grid can be identified by a one-,

two- or three-dimensional index that is accessible within the kernel through the built-in variable `blockIdx`. The size of the block can be determined through the built-in variable `blockDim`.

Thread blocks must execute independently, in parallel or serially. This requirement allows the scheduling of the thread blocks in any order and for any number of multiprocessors. Thus the programmers can write scalable code.

Threads within a block can cooperate by sharing data through shared memory and by synchronizing the execution in order to coordinate the memory access.

3. Memory hierarchy

CUDA threads can access data from different memory spaces during their execution, as depicted in Fig. 1.19.

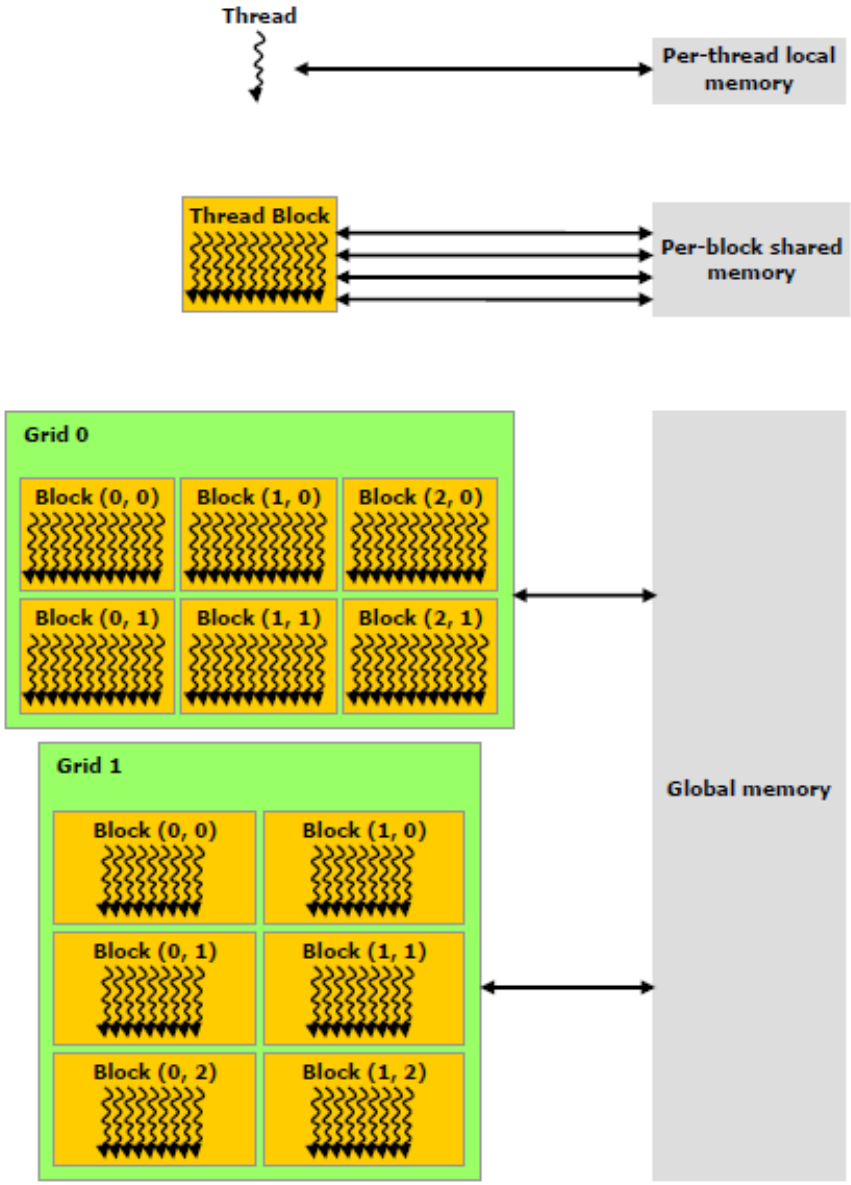

Fig. 1.19. Memory hierarchy (source: [28])

- Every thread has a private local memory and a register memory;
- Every thread block has a shared memory, which is visible by all the threads in the block. The shared memory has the same life span as the block;
- All the threads have access to the same global memory.

There are also two read-only memory spaces that are accessible by all the threads: the constant and the texture memory spaces. The global, constant and texture memory spaces are persistent across kernel launches by the same application.

4. Synchronization barriers

Programmers can specify synchronization points within a kernel by calling the `_syncthreads()` function. This function acts as a barrier where threads in a block wait for all the other to finish before proceeding with the next computation.

1.6. CONCLUSIONS

Medical image analysis and visualization represents a very challenging research field. There are a lot of methods that extract features from medical images and allow experts to visualize them. However, the data acquisition process is not perfect. The poor quality of the images represents a drawback for any medical image application. Image processing methods have to take into account all the imperfections and characteristics of medical data in order to generate results comparable to those obtained by manual handling.

Another drawback of medical applications is the large size of the processed data that leads to large computing times. This problem can be solved with the use of the GPGPU technology which enables programmers to create parallel applications that are considerably faster than those implemented on the CPU.

The next chapters propose fast automatic and semi-automatic methods that optimize the clinical workflow by allowing domain experts to investigate medical data, plan surgeries and follow the evolution of the patients in hip arthroplasty and angiology.

CHAPTER 2

RADIOGRAPHIC IMAGE ANALYSIS IN HIP ARTHROPLASTY

The results presented in this chapter were (or will be) published in the following papers:

A. Morar,F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "Computer Assisted Analysis of Orthopedic Radiographic Images", Proceedings of the 9th WSEAS International Conference on SIGNAL PROCESSING, pp. 66-71, Catania, 2010.

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "Medical Image Processing in Hip Arthroplasty", WSEAS TRANSACTIONS on SIGNAL PROCESSING, Vol. 6, Issue 4, pp. 165-174, 2010.

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "Multi-GPGPU Based Medical Image Processing in Hip Replacement", accepted for publication in the Journal of Control Engineering and Applied Informatics (to appear in no. 3, 2012).

This chapter describes an original algorithm for automatic/semi-automatic measurements and feature extraction in hip arthroplasty from radiographic images. The main idea behind the proposed method is based on the particularity that some parts of bones located at the level of the hip can be approximated by simple curves like lines or circles.

The next sections introduce the parameters that are important in hip arthroplasty and describe the steps of the proposed algorithm. They also give some implementation details, with the focus on the GPGPU paradigm for parallel processing.

2.1. PARAMETERS OF INTEREST IN HIP ARTHROPLASTY

The parts of bones that are of interest in hip replacement are the head, the neck and the body of the femoral bone, as well as the pelvic bone. Fig. 2.1 depicts the parameters extracted from an anterior-posterior radiographic image.

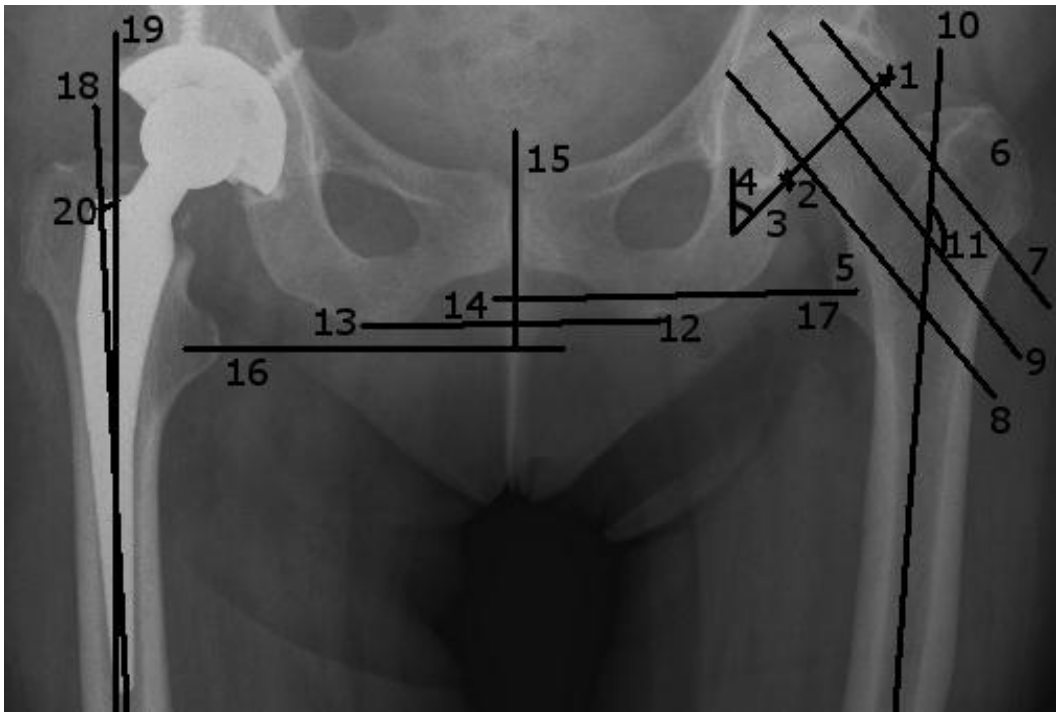

Fig. 2.1. Parameters important in hip arthroplasty extracted from an anterior-posterior orthopedic radiographic image

The parameters of interest in hip arthroplasty are slightly described in order to provide some insight into the hip replacement process:

1. The superior margin of the acetabulum - the superior point where the femoral bone, also known as the thigh bone, meets the pelvis;
2. The inferior margin of the acetabulum - the inferior point where the thigh bone meets the pelvis;

3. The femoral head axis or the acetabulum axis - the line connecting the superior margin and the inferior margin of the acetabulum;
4. The angle defined by the acetabulum axis and the vertical line;
5. The lesser trochanter;
6. The greater trochanter;
7. The tangent to the superior cortical of the femoral neck;
8. The tangent to the inferior cortical of the femoral neck;
9. The femoral neck axis - the axis of the cylinder determined by the tangents 7 and 8;
10. The femoral body axis - the axis of the cylinder that approximates the femoral body;
11. The cervico-diaphyseal angle, which is the most important parameter extracted from the radiographic image before the insertion of the prosthesis. It is the angle defined by the neck axis and the diaphyseal axis. The value of this angle determines whether a surgical intervention is necessary. If the angle ranges between 125 and 135 degrees, the thigh bone is considered to be healthy;
12. The right ischiadic tuberosity - the lowest right part of the pelvic bone;
13. The left ischiadic tuberosity - the lowest left part of the pelvic bone;
14. The ischiadic line or the horizontal reference line - the line that connects the two ischiadic tuberosities;
15. The vertical line which is perpendicular to the ischiadic line;
16. The line starting from the center of the lesser left trochanter, parallel to the ischiadic line;
17. The line starting from the center of the lesser right trochanter, parallel to the ischiadic line. The distance between lines 16 and 17 represents the vertical distance between the two femoral bones;

After the surgical intervention, there are other parameters that must be taken into account:

18. The diaphyseal axis of the femoral bone (the same as parameter 10);
19. The diaphyseal axis of the prosthesis or the axis of the prosthesis' body;
20. The deviation of the prosthesis - the angle defined by lines 18 and 19;

The parameters important in hip arthroplasty extracted from an anterior-lateral radiographic image after the insertion of the prosthesis are highlighted in Fig. 2.2:

21. The axis of the prosthesis' neck;
22. The axis of the prosthesis' body;
23. The anteversion angle - the angle defined by axes 21 and 22. If the value of this angle does not range between 5 and 10 degrees there is a high probability of prosthesis dislocation.

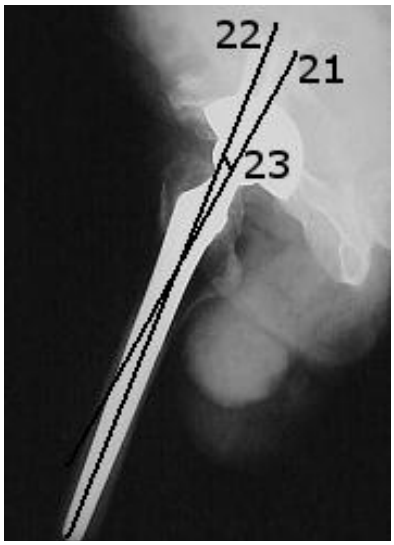

Fig. 2.2. Parameters important in hip replacement extracted from an anterior-lateral radiographic image, after the insertion of the prosthesis

2.2. AUTOMATIC/SEMI-AUTOMATIC MEASUREMENTS BASED ON CANNY EDGE DETECTION AND HOUGH TRANSFORM

This section describes an original method to automatically extract parameters from radiographic images in the hip arthroplasty field. It also provides user-interaction tools that can alter the automatic results.

The parameters extracted by the proposed method are: the ischiadic line, the reference vertical line, the vertical distance between the thigh bones, the deviation of the prosthesis, the center of the femoral head, the axis of the acetabulum, the angle defined by the axis of the acetabulum and the vertical line, the neck axis, the diaphyseal axis and the cervico-diaphyseal/anteversion angle. Fig. 2.3 depicts the flow of the algorithm.

In the first stage of the application, the contour of the bones is extracted with the Canny edge detector, as described in section 1.2.1. The output is a binary image, as illustrated in Fig. 1.6, where white pixels denote the contour, and black pixels, the background. The contour is disconnected in some parts of the X-ray and it cannot be reconstructed with accuracy. This is the reason we search for certain salient parameters that can be identified on partially extracted contours. Some of the salient parameters can be approximated by simple curves like lines, circles or ellipses.

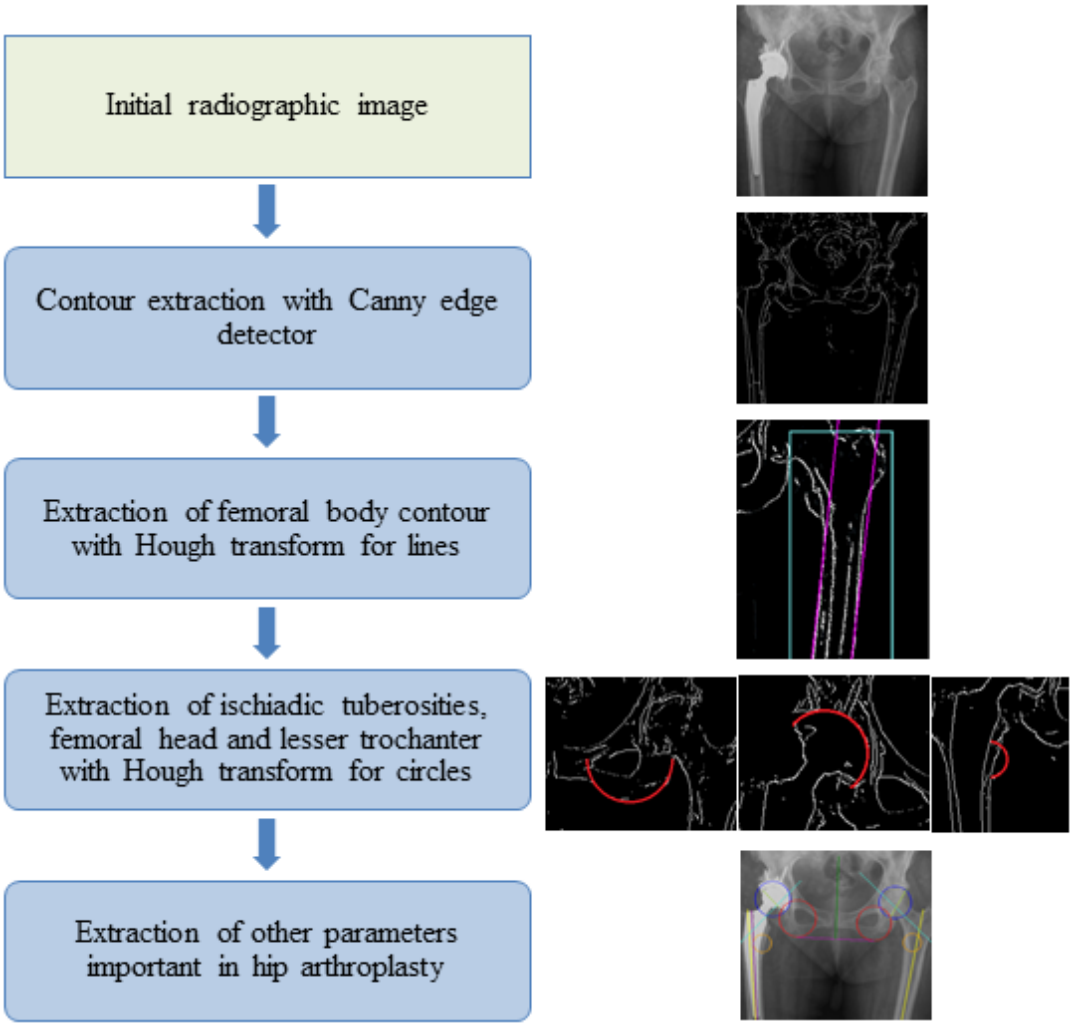

Fig. 2.3. The flow of the algorithm that automatically extracts the parameters important in hip arthroplasty

2.2.1. Detection of the Femoral Body

The first parameter extracted by the proposed algorithm is the contour of the femoral body. This step produces the most accurate results and does not depend on the extraction of the other parameters. The contour of the femoral body can be approximated by two lines, as depicted in Fig. 2.4. The Hough transform technique is used to extract the two lines that represent the contour of the femoral body.

The search space of the Hough transform for lines in an image is relatively large and leads to slow computing times. This space can be reduced based on the particularities of the radiographic images and the searched parameters. For instance, the femoral body is located in the lower half of the image and the lines that approximate its contour are nearly vertical.

Let w and h be the width, respectively the height of the radiographic image. The algorithm that detects the femoral body investigates only pixels with the y coordinate that satisfies the condition $y > h/2$. Also, the interval to search for θ is reduced to $[0,15] \cup [165,195] \cup [345,360]$.

This technique extracts straight lines that contain a number of contour pixels, or white pixels, greater than a given threshold T. The output is then refined and the most probable lines that approximate the contour of the femoral body are chosen. The algorithm is detailed below.

Fig. 2.4. Framing area for searching the lines that approximate the contour of the femoral body

Pseudo code 2.1. Detection of the femoral body

```
Apply Hough transform on a radiographic image
The output consists of a series of straight lines, all close to the area of the femoral body
for every pixel p of coordinates (x,y) where y>=h/2, do
        if p is white then (p is a contour pixel)
                for θ between 0 and 15, 165 and 195, 345 and 360, do (the discretization step is 1)
                        compute parameter r based on (1.21)
                        if r>1 and r<sqrt(w*w+h*h), then (r is within ranges)
                                increment accum[r,θ] (accum is the accumulator)
                        end if
                end for
        end if
end for
for every θ, do
        for every r, do
                if accum[r,θ]>T, then (T is a given theshold)
                        (r,θ) defines a line
                        Determine end-points of line (r,θ)
```

```
                    determine point (x', h/2) that contains the line (r,θ) (with (1.21))
                    determine point (x'', h) that contains the line (r,θ) (with (1.21))
                end if
            end for
        end for
        Determine the framing area for the lines of the femoral body contour
        determine the leftmost point (x_min1,h) of all the lines extracted with the Hough transform
        determine the rightmost point (x_max1,h) of all the lines extracted with the Hough transform
        determine the leftmost point (x_min2,h/2) of all the lines extracted with the Hough transform
        determine the rightmost point (x_max2,h/2) of all the lines extracted with the Hough transform
        x_min=min(x_min1,x_min2)
        x_max=max(x_max1,x_max2)
        Extract the lines that approximate the femoral body contour
        for y = h/2 to h, do
            find the leftmost white point in the interval [x_min,x_max] (it belongs to the first line)
            find the rightmost white point in the interval [x_min,x_max] (it belongs to the second line)
        end for
        compute the average coordinates (x1,y1) of all the leftmost white pixels
        compute the average slope s1 of the lines determined by the leftmost white pixels
        compute the average coordinates (x2,y2) of all the rightmost white pixels
        compute the average slope s2 of the lines determined by the rightmost white pixels
        End of pseudo code
```

The line defined by the point (x_1, y_1) and the slope s_1 approximates the left contour of the femoral body, while the line defined by the point (x_2, y_2) and the slope s_2 approximates the right contour of the femoral body.

This algorithm produces very good results, due to the fact that the body of the thigh bone occupies a significant area of the radiographic image. The large number of pixels that belong to the contour of the femoral body and the simple shape that approximates it represent an advantage for the proposed method.

2.2.2. Detection of the Ischiadic Tuberosities, the Femoral Head and the Lesser Trochanter

The ischiadic tuberosities, the femoral head and the lesser trochanter can be approximated by circular arcs, as illustrated in Fig. 2.5. The Hough transform for circles which is detailed in section 1.2.2 can be applied for the extraction of these parameters.

The ischiadic tuberosities are the next parameters determined by the proposed method. The extraction algorithm is based on the following observations:

- The ischiadic tuberosities are located between the two femoral bones

- The lowest part of an ischiadic tuberosity can be approximated by a circular arc. The radius of the arc is smaller than the width of the femoral body.
- In the area between the two femoral bones, in a bottom-up search, the ischiadic tuberosities are the first encountered circular-like structures.

Fig. 2.5. Parts of the pelvis and the femur that can be approximated by circular arcs

The tuberosity extraction algorithm ends at the first encountered circular arc. This method is very fast, due to the small number of white pixels located in the lower half of the radiographic image, between the femoral bones.

The ischiadic tuberosity does not have a perfect circular shape. If the radius of the circular arc is not considered a constant, but a variable that ranges between $R - \Delta R$ and $R + \Delta R$, where $\Delta R << R$, there is a bigger probability to correctly identify this parameter.

The algorithm does not search for whole circles, but for lower semicircles. Thus, the *y* coordinate of the pixels that belong to the circular arc have to be greater than the *y* coordinate of the center of the circular arc, $y_{pixel} > y_{circle}$, as depicted in Fig. 2.6.

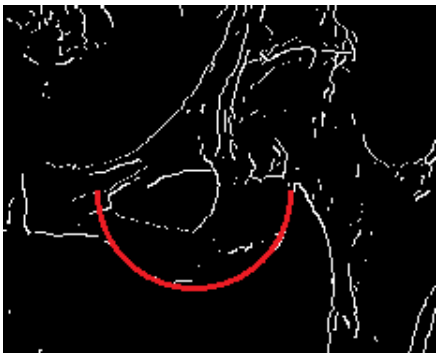

Fig. 2.6. Circular arc that approximates the ischiadic tuberosity

The tuberosity extraction algorithm is detailed in the following pseudo code:

Pseudo code 2.2. Detection of the ischiadic tuberosities

```
Determine the search area based on the previous extracted parameters, i.e., the femoral body contours
- x_min is the x coordinate of the rightmost pixel that belongs to the left femoral body contour
- x_max is the x coordinate of the leftmost pixel that belongs to the right femoral body contour
- y_min=h/2 and y_max=h where h is the height of the image
- w_body is the width of the femoral body
Apply modified Hough transform for circles on radiographic image
Search for the left ischiadic tuberosity
for y=y_max to y_min, do (bottom-up search)
        for x=x_min to w/2, do (the left tuberosity is located in the left half of the X-ray)
                if p(x,y) is white then
                        for R'=0 to w_body, do (the radius is smaller than the width of the femoral
                                                body)
                                for R=R'-ΔR to R'+ΔR, do (R is not constant)
                                        for b=y to y+R, do (look only for the lower semicircle)
                                        compute a based on (1.25)
                                                if x>=a-R and x<=a+R then
                                                        accum[a,b,R]++
                                                        Check if the semicircle has been found
                                                        If accum [a,b,R]>T, then (T is a given threshold)
                                                                break; (semicircle found)

                                                        end if
                                                end if
                                        end for
                                        Break if semicircle found
                                end for
                        end for
                        Break if semicircle found
                end if
        end for
        Break if semicircle found
end for
Search for the right ischiadic tuberosity
...
End of pseudo code
```

The femoral head extraction algorithm is based on the same idea as the previous one. The area to search for the center of the femoral head is set based on the previously extracted parameters. On the *x* axis we can observe that the femoral head is located between the ischiadic tuberosity and the diaphyseal axis. On the *y* axis, the femoral head is located above the ischiadic tuberosity. The left femoral head describes a circular arc with the following properties:

- the radius is not a constant, but ranges between $R-\Delta R$ and $R+\Delta R$, where $\Delta R << R$.

- the pixels that belong to the circular arc have to meet one of the conditions:
  - $y_{pixel} > y_{center} - \frac{R \cdot \sqrt{2}}{2}$ and $x_{pixel} > x_{center}$, or
  - $y_{pixel} > y_{center}$ and $x_{pixel} > x_{center} - \frac{R \cdot \sqrt{2}}{2}$.

Fig. 2.7 illustrates a circular arc that approximates the head of a prosthesis inserted in the left femoral bone. After the detection of the circular arcs that contain a number of white pixels greater than a given threshold, the final circular arc is determined as follows:

- the final radius is the average radius of the determined circular arcs
- the position of the final circular arc's center is the average center of the determined circular arcs

Fig. 2.7. Circular arc that approximates the head of a prosthesis inserted in the left femoral head

The algorithm is described in the following pseudo code:

Pseudo code 2.3. The extraction of the femoral heads

```
Extraction of the left femoral head
Determine the limits of the area to search for the circular arcs:
- x_min = the x coordinate of the left diaphyseal axis for y=h/2
- x_max = the x coordinate of the center of the left ischiadic tuberosity
- y_max = the y coordinate of the center of the left ichiadic tuberosity (the same as y_tub)
- y_min=y_tub - delta_y where delta_y depends on the width of the image (delta_y=w/5)
- w_body = the width of the femoral body
Modified Hough transform:
for y=y_min to y_max, do
        for x=x_min to x_max, do
                if p(x,y) is white then
                        for R'=0 to w_body, do
                                for R=R'-ΔR to R'+ΔR, do (R is not constant)
                                        for y=b-R to b+R, do
                                                compute x based on (1.25)
                                                if ((a>x and b>y-R*sqrt(2)/2) or
```

```
                                        (b>y and a>x-R*sqrt(2)/2)), then
                                            accum[a,b,R]++
                                            if accum[a,b,R]>T, then (T is a given threshold)
                                                save circle (a,b,R)
                                            end if
                                        end if
                                    end for
                                end for
                            end for
                        end if
                    end for
end for
Get circle (a_avg,b_avg,R_avg) where (a_avg,b_avg) is the average center computed based on all encountered circles, and R_avg is the average radius of the same circles
...
Extraction of the right femoral head
...
End of pseudo code
```

The lesser trochanter extraction algorithm is similar to the femoral head extraction method. Only the area where to search for the trochanter and the characteristics of the circular arc are modified. On the *y* axis, the limits where to search for the lesser trochanter are set to $[y_{tub} - \Delta y_{tub}, y_{tub} + R_{tub} + \Delta y_{tub}]$, where y_{tub} is the *y* coordinate of the center of the circular arc that approximates the ischiadic tuberosity and R_{tub}, its radius. On the *x* axis, the limits are $x_{head} - \Delta x_{head}$ and $x_{head} + \Delta x_{head}$, where x_{head} is the *x* coordinate of the center of the circular arc that approximates the femoral head. We set the parameters $\Delta y_{tub} = h/10$ and $\Delta x_{head} = h/10$. The pixels of the circular arc that approximates the left lesser trochanter have the following characteristic: the value of the *x* coordinate is greater than the value of the center's *x* coordinate, $x_{center} < x_{circle}$.

Fig. 2.8 illustrates a circular arc that approximates the left lesser trochanter.

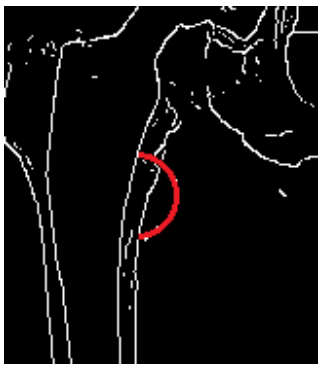

Fig. 2.8. Circular arc that approximates the left lesser trochanter

2.2.3. Extraction of other parameters of interest in hip arthroplasty

After the extraction of some of the parameters that can be approximated by lines and circles, the other parameters are automatically determined.

The ischiadic line connects the lowest points of the circular arcs that approximate the ischiadic tuberosities. After the extraction of the circular arcs, the points with the greatest *y* coordinate, one point belonging to an arc and one point to the other one, are extracted. The line that connects these two points is the ischiadic line.

The vertical line is perpendicular to the ischiadic line, in its middle.

The vertical distance between the two thigh bones depends on the ischiadic line, the vertical line and the two lesser trochanters. The application determines the lines that start from the centers of the two lesser trochanters, and are parallel to the ischiadic line. The vertical distance between the two thigh bones is computed as the length of the segment that connects the intersection points of the reference vertical line with the two parallel lines previously determined.

The diaphyseal axis depends on the two lines that represent the contour of the femoral body. If each femoral contour line is determined by two points, one with the *y* coordinate equal to the image height, $y = h$, and the other one with $y = h/2$, the diaphyseal axis is also determined by two points. The first point represents the average of the two points where $y = h$, one on the first line and one on the second line. The second point represents the average of the two points where $y = h/2$, one on the first line and one on the second line. The axis of the prosthesis' body is determined in a similar manner as compared to the extraction of the femoral diaphyseal axis.

The femoral neck axis is determined automatically, after the extraction of the femoral head. The inferior and superior corticals of the neck can also be approximated by circular arcs. The method follows the same modified Hough algorithm as the one that extracts the femoral head. The only changes are the coordinates of the framing areas where to search for the centers of the circular arcs and the characteristics of the circular arcs. We don't get further into details, because the algorithm is similar to the ones previously described. Fig. 2.9 illustrates the circular arcs that approximate the inferior and the superior cortical of the neck. The neck axis is the axis of the cylinder defined by the tangents to the inferior and superior cortical of the femoral neck.

Fig. 2.9. Circular arcs that approximate the inferior and superior cortical of the femoral neck

The femoral head axis connects the points where the femoral bone meets the pelvis. The inferior point where the femur meets the pelvis is determined based on three circles, as depicted in Fig. 2.10: the circle that approximates the ischiadic tuberosity, the circle that approximates the femoral head and the one that represents the inferior cortical of the femoral neck. The algorithm first finds the intersection of each circle with the other two. The result consists of three intersection points. The average of these points represents the inferior margin of the acetabulum. The superior point is also determined based on three circles, as illustrated in Fig. 2.10: the circle that approximates the femoral head, the circle that approximates the superior cortical of the femoral head, and the circular arc that approximates the region above the femoral head that is part of the pelvic bone contour. The average of the intersection points between each two circles represents the superior margin of the acetabulum.

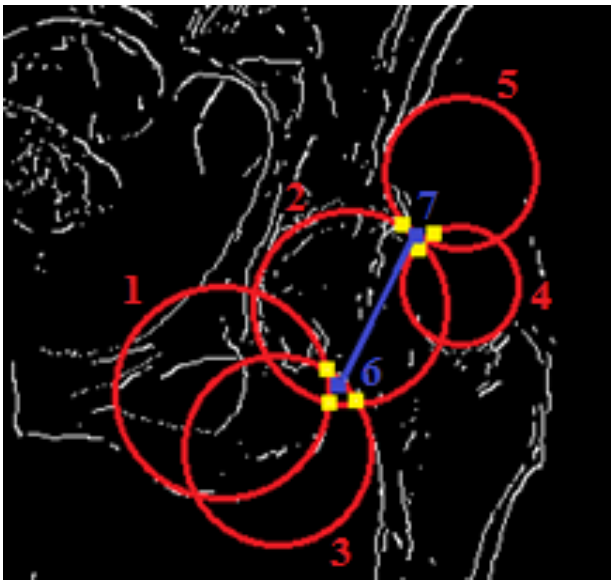

Fig. 2.10. Detection of the femoral head axis. The inferior point (6) where the pelvis meets the bone is the average of intersections (yellow points) between three circles: the circle that approximates the femoral head (2), the circle that approximates the ischiadic tuberosity (1) and the circle that approximates the inferior cortical of the femoral neck (3). The superior point (7) is the average of intersections (yellow points) between three circles: the circle that approximates the femoral head (2), the circle that approximates the superior cortical of the femoral neck (4) and the circle that approximates the pelvis in the location superior to the femoral head (5)

The angle between the acetabulum axis and the reference vertical line is computed automatically, after the extraction of the two lines that define it.

The cervico-diaphyseal/anteversion angle is computed automatically. It is the angle between the neck axis and the diaphyseal axis.

2.2.4. Results

The techniques that automatically extract parameters of interest from radiographic images were used to develop a software application whose aim was to reduce the time spent for the preparation of surgeries in hip arthroplasty. The application was part of the national project SABIMAS (Sistem informatic avansat, bazat pe imagistica medicala, pentru producerea implanturilor personalizate dedicata artroplastiei de sold - Advanced Information system, based on medical imaging, for personalized implants production in hip arthroplasty [39]).

Fig. 2.11 represents a screenshot of the application after all the described parameters have been identified. The application is considered semi-automatic due to the possibility of human intervention. In case the automatic method does not extract correctly some of the parameters, the user can define or alter straight segments and circles that approximate them.

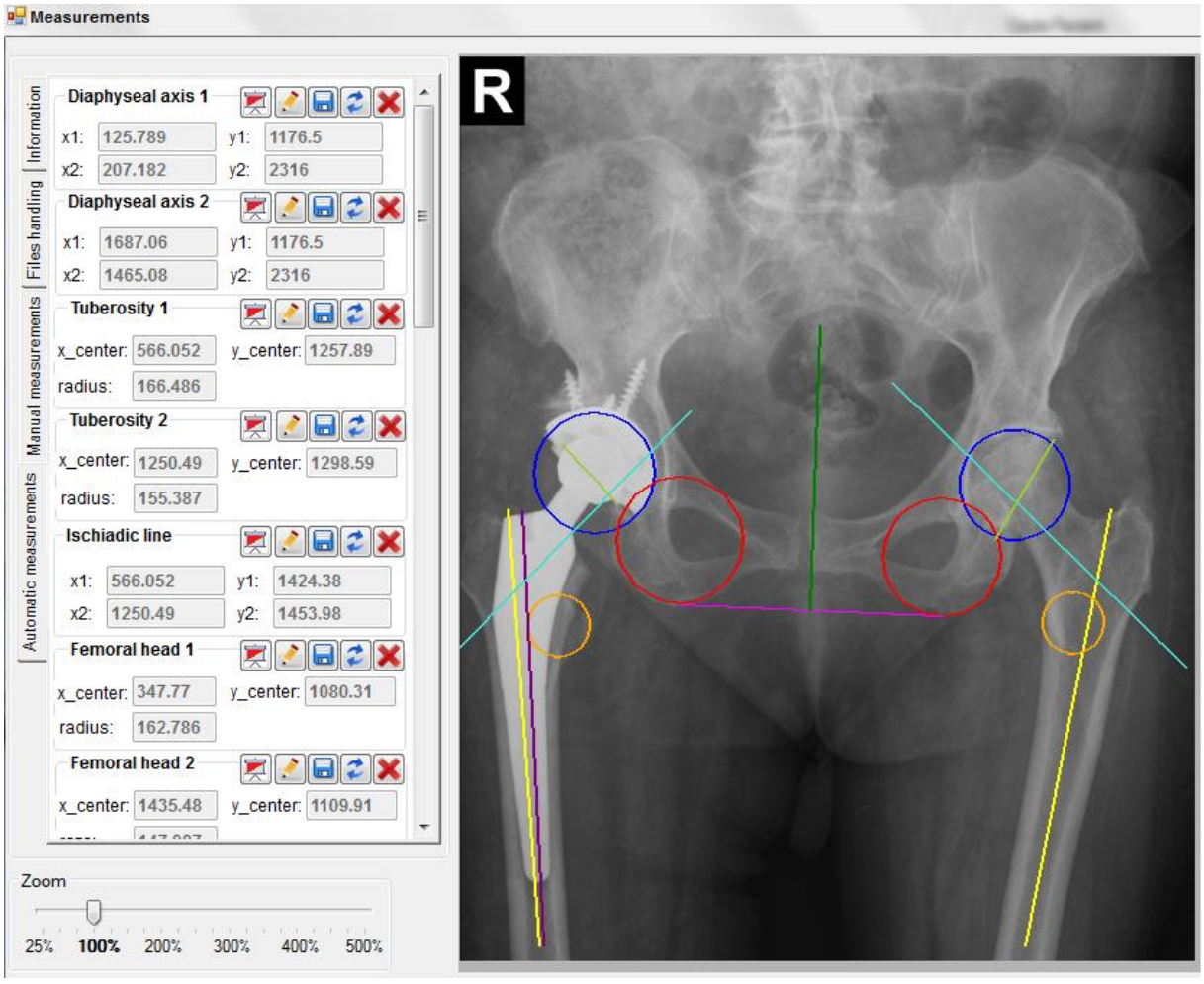

Fig. 2.11. Screenshot of the application that extracts the parameters important in hip arthroplasty

The application was designed after discussing with orthopedic specialists at Floreasca Emergency Clinical Hospital in Bucharest. After using the application, they declared that it

reduced the time spent for the extraction of parameters that are important in hip arthroplasty from approximately 30 minutes to an average of 2 minutes. The automatic extraction takes only a few seconds, but in some cases, not all the parameters are extracted correctly. In these cases, there is a possibility to manually adjust the wrongly determined parameters.

The radiographic images were processed both manually and automatically.

The automatic detection of the lines representing the femoral body contour was as accurate as the manual detection because of the large number of pixels that belong to the contour of the femoral body.

The automatic extraction of circles representing the ischiadic tuberosities led to a precision of approximately 90%. The errors were caused by the poor contrast between bones and other tissues in the area of the ischiadic tuberosities.

The automatic extraction of the circles representing the femoral heads is 85% accurate as compared to the manual one. Another cause for errors is the existence of other circular-like structures in the area of the femoral head.

The detection of the circles approximating the lesser trochanters did not generate very good results. The small number of pixels belonging to the contour of the lesser trochanters led to a precision of approximately 50%.

2.3. PARALLEL IMPLEMENTATION OF CANNY EDGE DETECTOR AND HOUGH TRANSFORM WITH CUDA

The large number of pixels in a radiographic image leads to slow computing times for feature extraction algorithms such as Canny edge detector and Hough transform. These computing times can be considerably reduced through the use of the GPGPU paradigm. CUDA implementations of these algorithms have already been developed. The novelty of our approach is in taking advantage of the processing power of multi-GPGPU systems and other CUDA architecture mechanisms. The particularities of orthopedic radiographic images are also taken into account to further reduce the computing times. This section gives details about the new implementations of the Canny edge detector and the Hough transforms and presents a comparison to other CPU and GPU implementations. All the tests regarding the computing times of the algorithms were made on an i7-2600K 3.40 GHz processor with 8GB RAM and two Nvidia GeForce GTX 590 GPU cards with 1.5 GB RAM.

2.3.1. Parallel Implementation of the Canny Edge Detector

The Canny edge detection method, proposed by Canny [8], is detailed in section 1.2.1. After many experiments, we decided that the last step in the Canny edge detector, namely the hysteresis thresholding, does not lead to significant improvements in finding the contour of bones in radiographic images. Also, the hysteresis thresholding step is quite difficult to implement in CUDA and time consuming. Therefore, the last step of the Canny edge detector

has been replaced in our implementation by a division of the image pixels into background and contour pixels based on a single threshold T.

Parts of the Canny edge detector have been successfully implemented on the GPU with Nvidia's Cg shading language by Fung [30]. Only the last part of the filter, i.e., the hysteresis thresholding, was not approached in his paper. Luo et al. [31] developed a new implementation of the Canny edge detector, taking advantage of the GPGPU technology. It follows all the steps of the algorithm, including the hysteresis thresholding.

2.3.1.1. Classic Canny Edge Detector with CUDA

This section presents a simplified implementation of the algorithm, which is based on the method proposed by Luo et al. [31] and the parallel implementation of the Sobel filter from the CUDA SDK Example [29].

The parallel implementation uses three kernels that are executed for all the pixels in the image. Before these kernels are run, the intensities of the image pixels are copied to the GPU texture memory.

The first step of the detector, the Gaussian filtering, is defined in one CUDA kernel. The kernel reads the intensities of the pixels located in a 3x3 neighborhood of the current pixel and updates the intensity of that pixel based on a Gaussian mask. Each thread block handles an image row, as illustrated in Fig. 2.12.

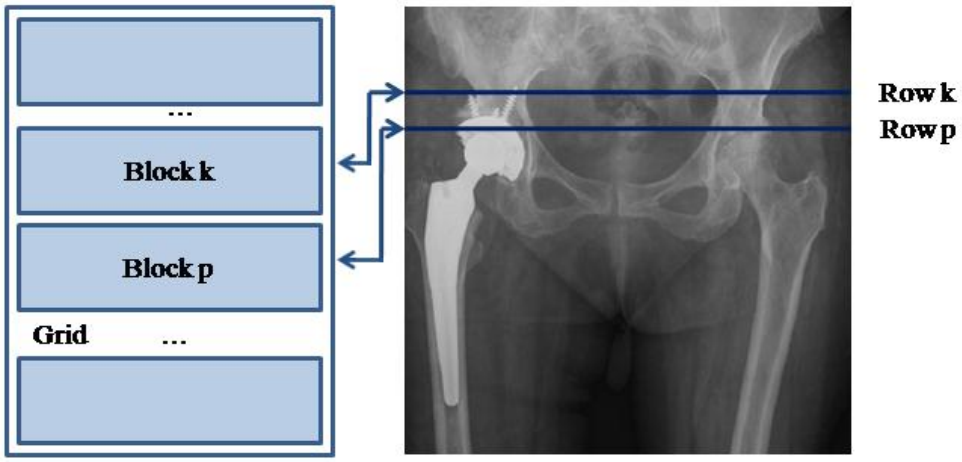

Fig. 2.12. Each block of threads handles one row in the image

The possibility of using shared memory has been explored, and is slightly described here. Each thread block copies parts of the image by loading data from texture memory to shared memory. Through barrier synchronization, each thread waits until the other threads have finished loading the corresponding data from texture to shared memory. Using the faster access to the shared memory, a thread updates in one iteration four pixels instead of one (as in Fig. 2.13). Again, after updating the pixels, the threads of a block synchronize.

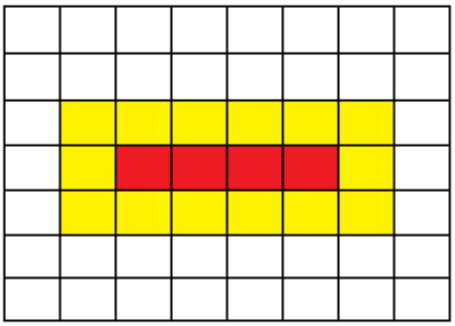

Fig. 2.13. In one iteration, a thread updates four pixels (the red pixels), based on their neighbors (the yellow pixels).

Table 2.1 shows the computing times of applying the Gaussian filter on images of different sizes, using shared and/or texture memory. The values represent the average of 10 tests for each image.

Table 2.1. GPGPU computing times for Gaussian filtering, with and without shared memory

Image size	Gaussian filtering using texture memory (ms)	Gaussian filtering using shared memory (ms)
256^2	0.202	0.218
512^2	0.381	0.363
1024^2	1.042	0.998
2048^2	3.543	3.172

Although the access to the shared memory is faster than the access to texture memory, the transfer between these two memory spaces leads to a lag and determines an insignificant difference between the two implementations for images of size up to 2048^2.

For an easier understanding of the parallel implementation of the Canny edge detector, the next steps of the algorithm are described using only texture memory.

The next three steps of the Canny edge detector, i.e., the computation of the gradient magnitude with the Sobel convolution kernels, the computation of the gradient direction and the discretization of the gradient direction, are defined in a single CUDA kernel. Within the kernel, the pixels in a 3×3 neighborhood of the current pixel are read from texture memory. For each pixel, the values of D_x and D_y are determined, the gradient's amplitude and the gradient's direction are computed using the Sobel operator. Then the gradient direction is discretized by using equation (1.20). Similar to the Gaussian filtering, each thread block handles an image row.

The last two steps of the Canny edge detector can be defined in a single CUDA kernel. Within the kernel, the discretized direction of the gradient and the intensity of the current pixel are read. The intensities of the pixel's neighbors on the gradient direction are determined. If the intensity of a neighbor is greater than the intensity of the current pixel, this pixel is removed

from the contour. The modified sixth step uses a single threshold T. All the pixels with intensity value greater than T are considered contour pixels. All the others are labeled as background pixels. Each thread block handles one image row.

The pseudo code of the CUDA Canny edge detector is given below.

Pseudo code 2.4. Canny edge detector with CUDA

```
Stage 1. Gaussian filtering
copy image from CPU to texture memory on the GPU
for every image pixel p, do (in parallel using a CUDA kernel)
        read values of p from a 3x3 neighborhood
        compute intensity based on the Gaussian mask
end for
copy output of Stage 1 into texture memory
Stage 2. Gradient magnitude and direction + discretization of gradient direction
for every pixel p, do (in parallel using a CUDA kernel)
        read pixel values from a 3x3 neighborhood of p
        determine Dx and Dy (gradient estimates in the horizontal and vertical direction)
        compute gradient magnitude (with (1.17))
        compute gradient direction (with (1.18))
        discretize gradient direction (with (1.20))
end for
copy the gradient magnitudes into texture memory
Stage 3. Non-maximum suppression and thresholding
for every image pixel A, do (in parallel using a CUDA kernel)
        Non-maximum suppression
        read gradient direction (discretized)
        determine neighbor pixels C and B along gradient direction
        read values vA, vB and vC of A, B and C
        if vA<vB or vA<vC, then
                vA=0
        end if
        Thresholding
        if vA>T (T is a given threshold), then
                vA=255  (A is considered to be a border pixel)
        else
                vA=0  (A is considered to be a background pixel)
        end if
end for
copy final image from GPU to CPU
End of pseudo code
```

2.3.1.2. Proposed Improvements to the Canny Edge Detector

This section describes our main improvements to the Canny edge detector implementation with CUDA, which are based on the use of page locked memory and multiple GPU cards.

One of the features recently introduced in the CUDA architecture is "zero-copy". This feature refers to direct GPU (device) access to CPU (host) memory. Before this, a simple CUDA program was structured into three stages:

- Copy data from CPU to GPU
- Process data on the GPU
- Copy data back from the GPU to the CPU

Now, the GPU devices can have access to the data on the host, if this data resides into page locked memory. The data can be allocated on page locked memory via `cudaHostAlloc()`. The instruction `cudaHostGetDevicePointer()` passes back the device pointer corresponding to the mapped, pinned host buffer allocated by `cudaHostAlloc()`. This eliminates the transfers between device and host. However, the access to page locked memory is slower than the access to device memory. So, if a kernel needs to access the data frequently, a compromise should be made:

- Allocate data on CPU page locked memory
- Copy data from CPU to GPU
- Process data on the GPU
- Copy data back from GPU to page locked memory

This approach is faster than the normal flow in a CUDA program because the transfer between page locked memory and device memory is faster than the transfer between pageable memory and device memory. Even if it can reduce the memory transfer duration, page locked memory is limited and can also degrade system performance, since it reduces the amount of memory available to the system for paging.

We performed several experiments, each representing a different implementation of the Canny edge detector, in order to observe the effect of using various characteristics of the CUDA architecture:

- The first proposed improvement to the CUDA implementation of the Canny edge detector takes advantage of the “zero-copy” feature. The data, i.e., the radiographic image, is allocated on page locked memory with `cudaHostAlloc()`. A pointer to this data that can be accessed by the device is obtained with the instruction `cudaHostGetDevicePointer()`. The remainder of the algorithm is similar to the one in pseudo code 2.4, without the last instruction that copies the data from the GPU back to the CPU.

- The second improved version uses page locked memory, but still copies the data between CPU and GPU before and after applying the Canny edge detector.

- The last proposed version of the Canny edge detector with CUDA offers the possibility to run the algorithm on multiple GPU cards. If a problem can be divided into several sub-

problems to be solved independently on different GPUs, the computing times can be significantly reduced. Let N be the number of available graphic cards on a computer. The initial image can be divided on the y axis into N sub-images that are processed independently, as illustrated in Fig. 2.14.

Let w and h be the width, respectively the height of the radiographic image. We can assume without loss of generality that h is a multiple of N.

The output of each GPU is a sub-image of width w and height h/N. The output image of GPU_k starts with the $[(k-1)\cdot h/N+1]^{th}$ row and ends with the $(k\cdot h/N)^{th}$ row. Each pixel is processed based on the pixels from a 3×3 neighborhood. Thus, for the correct processing of the pixels located at the borders of the sub-images, the input of each GPU is a sub-image of width w and height $h/N+2$. The input sub-image of GPU_k starts with the $[(k-1)\cdot h/N]^{th}$ row and ends with the $(k\cdot h/N+1)^{th}$ row. Exceptions are the first and the last GPU, which have input images of height $h/N+1$, as can be observed in Fig. 2.14.

Data can be processed on different GPUs concurrently through streams, as exemplified in the "simpleMultiGPU" application from the CUDA SDK example [29]. The sub-images are copied from page locked memory on the CPU to GPU and back asynchronously with cudaMemcpyAsync().

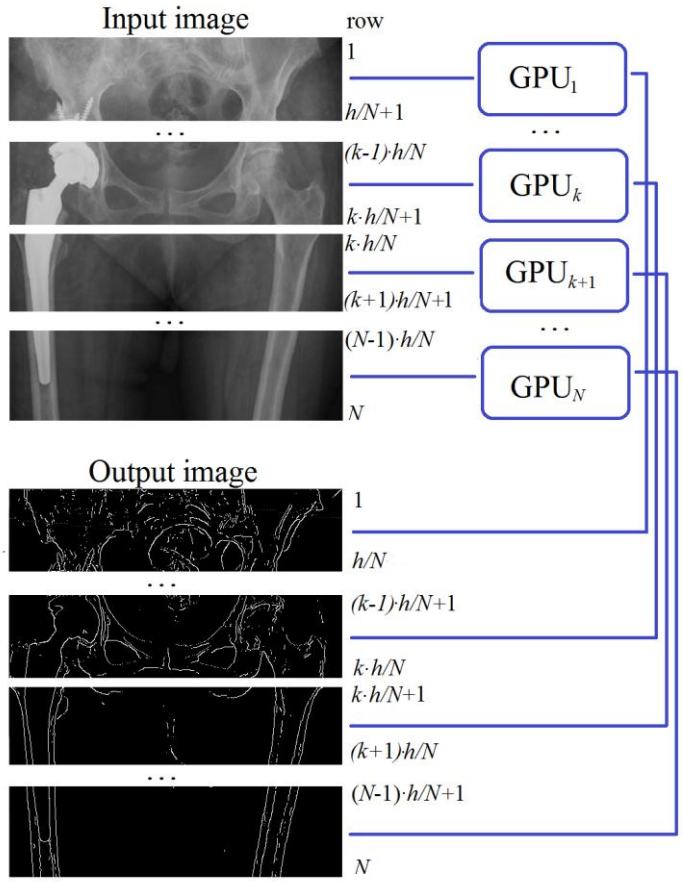

Fig. 2.14. Division of input and output images for multi-GPGPU processing

Arithmetic operations on N GPUs are N times faster than the same arithmetic operations on a single GPU. However, the division of data and the memory transfers between the CPU and multiple GPUs introduce a lag that can eliminate the multi-GPGPU performance gain. This issue is further discussed in the Results section 2.3.3.

The pseudo-code of the multi-GPGPU Canny edge detector is given below.

Pseudo code 2.5. Multi-GPGPU Canny edge detector

```
Divide the data according to the number of available GPUs in the system
for k=0 to N, do
        set active device GPUk
        allocate memory on CPU for sub-image k (with cudaHostAlloc())
        copy from initial image to input sub-image k on the CPU
        allocate memory for sub-image k on GPUk
        create stream for GPUk
end for
Copy data to/from each GPU asynchronously and apply Canny edge detector on each GPU
for k=0 to N, do
        set active device GPUk
        copy input data from CPU to GPUk (with cudaMemcpyAsync())
        apply Canny on sub-image k (in parallel using CUDA kernels that run for the current stream)
        copy output data from GPU k to CPU (with cudaMemcpyAsync())
end for
Synchronize streams
for k=0 to N, do
        set device GPUk
        wait for all operations of stream k to finish
        delete stream k
        copy data from output sub-image on the CPU to final image on the CPU
end for
End of pseudo code
```

The improvements obtained by the several experiments described in this section are reflected in the comparison regarding the computing times of different implementations of the Canny edge detector that are presented in section 2.3.3.

2.3.2. Parallel Implementations of the Hough Transforms

The techniques that extract lines and circles from binary images, i.e., the Hough transform for lines and the Hough transform for circles, are described in sections 1.2.2.1 and 1.2.2.2 respectively.

In their paper, Braak et al. [32] present three GPGPU implementations of the Hough transform for lines that reduce the computing times considerably. Other research directions

presented by Chen et al. [41] follow ways of accelerating the Hough transform both for lines and circles.

This section describes intuitive CUDA implementations of the Hough transforms and proposes new methods that use multiple GPU cards and focus on the particularities of the processed images and the searched parameters.

2.3.2.1. CUDA Implementation of the Hough Transform for Lines

The intuitive implementation of the Hough transform for lines described by Braak et al. [32] follows the CPU implementation of the algorithm. However, in the parallel algorithm, all threads have concurrent access to data. In order to avoid memory read/write hazards, the incrementing of the accumulator at location (r,θ) has to be done using an atomic operation, `atomicAdd()`. This operation serializes the access of the threads to the accumulator, representing at a first glance an inefficient use of the parallel architecture.

The updating of the accumulator is defined in a CUDA kernel that is executed for every image pixel. The intensity of the current pixel is read from a binary image, where the important pixels are white and the background pixels are black. In the proposed radiographic image processing method, the binary image is the output of the Canny edge detector. If the current pixel is white, the algorithm searches for all the candidate lines that contain that pixel. For all θ, the second parameter r is computed by using (1.21). If $r \in \left[1, \sqrt{w^2+h^2}^2\right]$, the accumulator at location (r,θ) is incremented. Every thread block handles an image row.

According to Braak et al. (2011), the atomic operation that updates the accumulator introduces a considerable lag. We introduced a test in order to observe if the atomic operations lead to slow computing times for the particular case of feature extraction from orthopedic radiographic images. We compared two implementations, one using atomic operations for incrementing the accumulator, and one that does not use atomic operations. The second implementation that simply increments the accumulator without `atomicAdd()` leads to wrong results, and it is not considered an alternative to the first method. It is used only to observe the effect of atomic operations on the speed of the algorithm. Table 2.2 presents the computing times of these two implementations, for radiographic images of different sizes. The values represent the average of 10 tests for each image. As can be observed, the differences between the two implementations are insignificant. Therefore we can conclude that, for radiographic images, where the contour pixels represent less than 3% of the total number of image pixels and the straight lines are very sparse, the atomic operation does not affect the performance of the CUDA Hough transform.

Table 2.2. GPGPU computing times of Hough transform for lines with atomic/without atomic operations

Image size	Hough transform for lines with atomicAdd() (ms)	Hough transform for lines without atomicAdd() (ms)
256^2	3.115	3.111
512^2	7.841	7.91
1024^2	19.767	19.818
2048^2	54.478	54.185

The pseudo code of the CUDA Hough transform for lines is described below.

Pseudo code 2.6. Hough transform for lines

```
transfer image data from CPU to GPU texture memory
Initialize accumulator
for every pair (r,θ), do (in parallel using a CUDA kernel)
        accum[r,θ]=0
end for
Update accumulator
for every pixel p, do (in parallel using a CUDA kernel)
        if p is white, then
                get x and y coordinates of p (based on the index of the CUDA thread)
                for θ = 0 to 360, do (discretization step is 1)
                        compute r
                        if r>1 and r<sqrt(w*w+h*h), then
                                increment accumulator accum at location [r,θ] (with atomicAdd())
                        end if
                end for
        end if
end for
Determine the lines in the image based on the value of the accumulator
for every θ, do
        for every r, do
                if accum[r,θ]>T, then (T is a threshold)
                        (r,θ) is a line
                end if
        end for
end for
End of pseudo code
```

2.3.2.2. Proposed Improvements to the Hough Transform for Lines

This section describes our experiments whose goal was to observe the effect of using different characteristics of the CUDA architecture or taking advantage of particularities of the investigated images.

- The first change to the previously described implementation is to allocate page locked memory for the image data with `cudaHostAlloc()`. If the CPU data is accessed directly by the device with `cudaHostGetDevicePointer()`, the algorithm is slower than the initial one. The reason is the frequent access of the GPU to the accumulator.

- The other version that uses page locked memory without eliminating the copies between CPU and GPU is a faster alternative.

- The next experiment that alters the initial implementation of the Hough transform for lines is a multi-GPGPU approach. It divides the image in a similar way as compared to the one described for the multi-GPGPU Canny edge detection method. The input and output sub-images of GPU_k for the Hough transform for lines have the same size as the output sub-image of GPU_k for the Canny edge detector. Here there is no need to duplicate the border rows for the input sub-images because the accumulator is not processed based on neighboring pixels.

  If the data that represents the image can be divided based on the number of available GPUs, this cannot be accomplished for the accumulator. For each GPU, a sub-accumulator of the same size as the initial one is allocated on the CPU on page locked memory and on the current GPU. We use the following expression to build the final accumulator after the sub-accumulators of each GPU have been updated and the data has been copied back to the CPU:

$$accum(r,\theta) = \sum_{k=1}^{N} accum_{GPU_k}(r,\theta) \tag{2.1}$$

- We performed another experiment that changes the structure of the CUDA grids and blocks.

  In the classic implementation, the threads are structured into one-dimensional blocks and the blocks are structured into a one-dimensional grid. The size of the grid is equal to the height of the image. The size of a block is set to 256. Each block handles a single image row. If the width of the image is greater than the number of threads in a block, a thread processes more than one pixel.

  The new approach structures the CUDA threads based on the size of the image, but also on the size of parameter θ. The blocks are structured into two-dimensional grids, with the width and height equal to the image width and image height respectively. The size of a block is equal to the number of possible values for parameter θ. Thus, a thread does not handle all the candidate straight lines that contain a pixel, but a single line defined by the parameter θ

that contains the current pixel. This implementation is faster than the first one if the search space, namely the size of the image and the size of parameters r and θ, is relatively small. Otherwise, there are not enough GPU cores to handle all the data concurrently, and the CUDA driver serializes the process. The lag introduced by this serialization can eliminate the performance gain obtained by the different structuring of the CUDA threads.

- The initial interval where to search for the parameter θ is [0,360]. If the particularities of the orthopedic radiographic images are taken into account, this interval can be significantly reduced, as discussed in section 2.2.1. For instance, if the application searches for straight lines that approximate the contour of the femoral body in images at the level of the hip, only the lines that are almost vertical should be considered. The search space for θ is reduced to the interval $[0,15]\cup[165,195]\cup[345,360]$. Hence, if the discretization step is 1, a block has 45 threads instead of 360, one thread for each value of θ.

- We also implemented a parallel version of the Hough transform for lines that uses the gradient's direction matrix to reduce the complexity of the algorithm form $O(k \cdot N)$ to $O(N)$, where N is the number of pixels in the input image and k is the number of discrete values of parameter θ in the search space. This method is further detailed in section 1.2.2.1.

A comparison between the existing and proposed implementations is described in section 2.3.3.

2.3.2.3. CUDA Implementation of the Hough Transform for Circles

This section presents the intuitive parallel implementation of the Hough transform for circles. We call this implementation intuitive because it follows the CPU implementation exactly, with the only difference that the CUDA method processes the image pixels in a parallel manner.

The accumulator is updated in a CUDA kernel that is executed for all the image pixels. If the current pixel is of interest, i.e., it is a white pixel, the radius R is computed for every pair (a,b) based on (1.25). If R is within range, the accumulator is incremented at location (a,b,R) with atomic operations. All the triplets (a,b,R) where the accumulator has the value greater than a given threshold T describe a circle.

Similar to the observation in section 2.3.2.1 regarding the computing times for the implementation of the Hough transform for lines with atomic operations, there is a small number of circles extracted from radiographic images. This is the reason why atomic operations are not considered to introduce a significant lag in this particular case.

The pseudo code of the CUDA Hough transform for circles is given below.

Pseudo code 2.7. Hough transform for circles

```
transfer image data from CPU to texture memory on the GPU
Initialize accumulator
for every triplet (a,b,R) do (in parallel using a CUDA kernel)
        accum[a,b,R]=0
end for
Update accumulator
for every pixel p, do (in parallel using a CUDA kernel)
        if p is white, then
                get x and y coordinates of p
                for a = 1 to w, do
                        for b = 1 to h, do
                                compute R (based on (1.25))
                                if R>1 and R<sqrt(w*w+h*h), then
                                increment accum[a,b,R] (with atomicAdd())
                                end if
                        end for
                end for
        end if
end for
for every a, do
        for every b, do
                for every R, do
                        if accum[a,b,R]>T, then (T is a given threshold)
                                (a,b,R) defines a circle
                        end if
                end for
        end for
end for
End of pseudo code
```

2.3.2.4. Proposed Improvements to the Hough Transform for Circles

This section describes our experiments to improve the computing times of the Hough algorithm for circles with CUDA. They are similar to the ones proposed in 2.3.2.2.

The parameter space of the Hough transform for circles is three-dimensional and occupies large amounts of memory. This represents an impediment for using page locked memory. For an image of size 512^2, the accumulator occupies $size(R) \cdot size(a) \cdot size(b) \cdot sizeof(\text{int}) = 724\,\text{MB}$ of memory. On the computer used for testing it was impossible to allocate such amount of pinned memory. Also, the direct access to the data

that resides on page locked memory via `cudaHostGetDevicePointer()` introduces a lag that reaches the maximum run-time for the kernel launches. These are the reasons we did not use the implementations that allocate page locked memory. Among these implementations there is also the multi-GPGPU method, which allocates page locked memory for the asynchronous transfers between the host and the multiple devices.

The size of the accumulator raises also a problem for allocating GPU memory. The accumulator associated to an image of size 1024^2 occupies more than 5.7 GB of memory, while the graphic card used for testing has only 1.5 GB of GPU RAM. There are two approaches that can avoid the problem of the GPU memory limitation.

- The first solution would be to divide the parameter space (a, b, R) into sub-spaces of smaller size that can be processed serially on the GPU. The intuitive implementation of the algorithm first investigates the image space (x, y) with one thread per pixel and continues with the parameter space (a, b, R) if the current pixel is white. Thus, in the intuitive approach a thread has to have access to the whole accumulator. The division of the parameter space can be accomplished only if the order of the search spaces is changed. Therefore, instead of first investigating the image pixels and then moving to the parameter space, the algorithm could initially search the parameter space with one thread per triplet (a, b, R) and then compute x based on the value of y. This approach allows the processing of larger data, but it is not practical. In the classic implementation, the parameter space is investigated only if the current pixel is of interest, while in the second approach the parameter space is investigated for all the pixels in the image space. This represents a waste of hardware resources, since only approximately 3% of the image pixels are of interest.

- The second solution offers the possibility to process larger data by reducing the parameter search space based on the particularities of the investigated images. For example, if the application searches for circles that approximate the ischiadic tuberosities, the lesser trochanters and the femoral heads in radiographic images, the radii of these circles can be set within certain intervals. The circles representing the ischiadic tuberosities and the femoral heads have radii smaller than the width of the femoral body; the circles approximating the lesser trochanters have a radius smaller than half the width of the femoral body. After determining the femoral body with the Hough transform for lines, the intervals for the radii of the circles can be set. Other constraints that further reduce the parameter space for the particular case of radiographic images in the hip arthroplasty field are described in section 2.2.2. We chose this approach to solve the problem of the memory limitation because it is easier to implement and faster than the first proposed solution.

- Another experiment that alters the original implementation of the Hough transform for circles is characterized by the different structuring of the CUDA threads. The new approach organizes the threads based on the size of the image, but also based on the size of the circle radius R. The blocks are structured into two-dimensional grids, with the width and height

equal to the image width and image height, respectively. The size of a block is equal to the number of possible values of parameter R. Hence, a thread does not handle all the possible straight circles that contain a pixel, but a single circle of radius R that contains the current pixel. The current thread computes for every $b \in [\max(0, y-R), \min(y+R, h)]$ the value of parameter a, based on (1.25). If a is within range, the accumulator is incremented at location (a, b, R). In the classic implementation, a and b loop through their whole search intervals. In this new implementation, the search space of b is reduced based on the radius R, while a is computed based on the value of b. The comparison between the computing times of the classic implementation and those of the proposed ones is described in the Results section 2.3.3.

2.3.3. Results

This section presents the computing times for some of the existing CPU and GPU implementations, as well as for the methods proposed in this thesis. The improvements obtained by our methods result either from using multiple GPU cards, by taking advantage of other features of the CUDA architecture, or by taking into account the particularities of the investigated images.

The CPU classic implementation of the Canny edge detector was compared to the implementation with CUDA from section 2.3.1.1 and the new implementations that were proposed in section 2.3.1.2. Table 2.3 shows the computing times in ms for each method applied on radiographic images of different sizes. Each value represents the average of 10 tests. The third column in the table represents the intuitive implementation of the Canny edge detector with CUDA which is described in section 2.3.1.1. The forth column presents the computing times for the implementation that uses the "zero-copy" feature of the CUDA architecture. The fifth column denotes the implementation that allocates page locked memory on the CPU, but also transfers the data to/from the GPU. The last column stands for the multi-GPGPU implementation of the Canny edge detector.

Table 2.3. Computing times for the Canny edge detector on the CPU and on the GPU

Image size (pixels)	CPU implem. (ms)	GPGPU classic implem. (ms)	GPGPU implem. with "zero-copy" (ms)	GPGPU implem. with pinned memory and CPU-GPU transfers (ms)	multi-GPGPU implem. (ms)
256^2	15	0.341	0.345	0.279	0.624
512^2	39	0.701	0.662	0.606	0.783
1024^2	153	2.149	1.931	1.867	1.392
2048^2	566	7.668	6.877	6.793	4.104

The GPGPU implementations introduce an impressive performance gain as compared to the classic CPU implementation of the Canny edge detector. For an image of size 256^2, the GPGPU classic implementation runs 44x faster than the CPU one. This difference increases with

the size of the image. For an image of size 2048^2, the classic GPGPU method, which leads to the slowest computing times on the GPU, runs 73x faster than the CPU implementation of the algorithm.

As can be observed, the implementation that allocates page locked memory on the CPU and transfers it via cudaMemcpy() to/from the GPU relatively improves the computing times as compared to the classic CUDA implementation. The method where the GPU accesses the data directly from page locked memory is comparable to the one that transfers data between pinned memory and GPU, but decreases performance for large images because of the frequent access of the GPU device to the memory. The fastest implementation, the multi-GPGPU one, reduces the computing times significantly. For an image of size 256^2 the lag introduced by the division of data into sub-images and their transfer to/from the GPU leads to slower computing times than for other implementations. But for an image of size 2048^2, two graphic cards run the Canny edge detector almost two times faster than a single graphic card.

Table 2.4 shows the computing times, representing the average of 10 tests for each image, for the implementations of the Hough transform for lines, both on the CPU and GPU. The first GPU implementation is the one described in section 2.3.2.1. The second and third GPU implementations allocate page locked memory with cudaHostAlloc(). In the second implementation the data is accessed directly by the GPU via cudaHostGetDevicePointer() and in the third one, the data is transferred to/from the GPU via cudaMemcpy(). The last column in the table shows the computing times for the multi-GPGPU implementation of the Hough transform for lines.

Table 2.4. Computing times of the intuitive Hough transform for lines on the CPU and on the GPU

Image size (pixels)	CPU implem. (ms)	GPGPU classic implem. (ms)	GPGPU "zero-copy" implem. (ms)	GPGPU implem with pinned mem. and CPU-GPU transfer (ms)	multi-GPGPU implem. (ms)
256^2	152	1.955	10.209	1.779	2.088
512^2	218	4.615	23.836	4.277	3.784
1024^2	1995	19.344	113.879	18.695	12.642
2048^2	9890	89.494	538.171	87.312	52.979

The difference in computing times between the CPU and the GPU implementations increases with the size of the processed data and the number of arithmetic operations. For the Hough transform, where the processed data consists of the image space, but also the parameter space, the performance gain introduced by the GPU is even greater than the one observed for the Canny edge detector. In case of an image of size 256^2, the classic GPGPU implementation runs 77x faster than the CPU one, while for an image of size 2048^2, the GPGPU implementation adds a performance gain of 110x. The “zero-copy” implementation is very slow because of the frequent device access to the page locked memory for the updating of the accumulator. The implementation that stores the accumulator on the CPU on page locked memory and transfers data to/from the GPU is faster than the other single-GPGPU implementations. But the real

performance gain can be observed for the multi-GPGPU implementation, that runs at least 1.6x faster than any other GPU implementation for a radiographic image of size 2048^2.

The next table shows the computing times for similar implementations of the Hough transform, as compared to the ones previously discussed. The only difference is the structuring of the CUDA threads. Instead of searching for all the lines that contain the current pixel, a thread searches for the line defined by a certain parameter θ that contains the current pixel.

Table 2.5. Computing times of the Hough transform for lines on the CPU and on the GPU - with the different structuring of the CUDA threads

Image size (pixels)	GPGPU classic implem. (ms)	GPGPU "zero-copy" implem. (ms)	GPGPU implem with pinned mem. and CPU-GPU transfer (ms)	multi-GPGPU implem. (ms)
256^2	3.184	11.201	3.028	2.715
512^2	7.995	20.181	7.536	6.305
1024^2	19.842	89.541	19.151	14.926
2048^2	54.907	439.531	52.502	38.875

The performance gain introduced by the different structuring of the CUDA threads can be observed only for very large images (2048^2).

Table 2.6 shows the computing times of the Hough transform for lines, if the search space of parameter θ is reduced from the interval $[0,360]$ to $[0,15] \cup [165,195] \cup [345,360]$.

Table 2.6. Computing times of the Hough transform for lines on the CPU and on the GPU - with reduced search space for parameter θ

Image size (pixels)	CPU implem. (ms)	GPGPU classic implem. (ms)	GPGPU "zero-copy" implem. (ms)	GPGPU implem with pinned mem. and CPU-GPU transfer (ms)	multi-GPGPU implem. (ms)
256^2	33	0.701	4.857	0.549	1.088
512^2	39	1.996	11.744	1.696	2.175
1024^2	337	7.475	55.912	6.553	5.732
2048^2	1656	29.272	240.334	27.235	18.647

Even if the different structuring of the CUDA threads can lead to faster implementations of the Hough transform for lines, the impressive performance gain is observed for the methods that reduce the parameter space. This is only an example of how the particularities of the inspected images can influence the computing times. Section 2.2.1 describes further ways to reduce the parameter search space for the particular case of radiographic image processing in hip replacement.

Table 2.7 shows the computing times of the last implementations of the Hough transform for lines, which remove the "for" loop of parameter θ based on the gradient's direction, as explained in section 2.3.2.2.

Table 2.7. Computing times of the Hough transform for lines on the CPU and on the GPU - with computation of θ based on the gradient direction

Image size (pixels)	CPU implem. (ms)	GPGPU classic implem. (ms)	GPGPU "zero-copy" implem. (ms)	GPGPU implem with pinned mem. and CPU-GPU transfer (ms)	multi-GPGPU implem. (ms)
256^2	2	0.365	0.447	0.181	1.069
512^2	2	0.645	0.994	0.389	1.949
1024^2	9	1.516	2.421	0.945	4.798
2048^2	41	4.311	8.609	2.851	13.86

This method leads to the best computing times. The third CUDA implementation that allocates page locked memory and transfers data between the CPU and the GPU is the fastest. The results show that the multi-GPGPU approach is not well suited for applications where the threads are not arithmetic intensive. The lag introduced by the division of data and memory transfer between the CPU and the devices eliminates any performance gain introduced by the use of multiple graphic cards.

Table 2.8 shows the computing times for the CPU and GPU implementations of the Hough transform for circles. The first GPU implementation is the one described in section 2.3.2.3. The second implementation changes the structure of the CUDA threads so that a thread does not handle all the circles that contain the current pixel, but only one circle of radius R that contains the current pixel. The third CUDA implementation shows the impact of reducing the search space of the radius from the interval $[1,\sqrt{w^2+h^2}]$ to $[1,\sqrt{w^2+h^2}/10]$, based on the properties of the bones in orthopedic X-rays, which are detailed in section 2.2.2. The first CPU implementation is the intuitive one, and the second one reduces the search parameter of R to $[1,\sqrt{w^2+h^2}/10]$.

Table 2.8. Computing times for the Hough transform for circles on the CPU and GPU

Image size (pixels)	CPU classic implem. (ms)	CPU implem. with reduced search space (ms)	GPGPU classic implem. (ms)	GPGPU implem. with different structuring of the CUDA threads (ms)	GPGPU implem. with reduced search space (ms)
256^2	11622	57	389.831	24.154	2.907
512^2	157275	453	2787.51	186.429	20.169
1024^2	1019555	4798	x	x	149.995

The GPU implementations that allocate memory for the whole accumulator on the device cannot handle images of size 1024, because of the memory limitation, as discussed in section 2.3.2.4. The implementation that structures the CUDA grid and blocks so that a thread handles all the circles that contain the current pixel is considerably slower than the implementation where a thread handles only one circle defined by the current pixel and a certain radius. If the search space for the radius is reduced according to the particularities of the radiographic images and the investigated parameters, the computing times are further improved.

Similar to the tests for the Hough transform for lines, the current tests represent only an example of how the reduction of the search space can influence the computing times. Section 2.2.2 describes further ways to reduce the search space in the particular case of orthopedic radiographic images in the hip arthroplasty field.

2.4. CONCLUSIONS

This chapter approaches automatic feature extraction methods in hip arthroplasty from radiographic images. The main idea behind the proposed techniques is based on the particularity that certain parts of bones in hip arthroplasty like the femoral body and the femoral head can be approximated by lines or circles. The main contributions of the chapter are shortly outlined.

- The first part of this chapter describes a practical contribution, which consists of an application for the automatic/semi-automatic extraction of parameters of interest in hip arthroplasty.

  The first stage of the parameter extraction method is the contour detection with the Canny algorithm. The poor contrast between bones and other tissues in radiographic images leads to a disconnected contour. This problem can be overcome with the use of the Hough transforms that detect imperfect and incomplete instances of circles and lines

  Hough transform for lines is used to detect straight lines that approximate the contour of the femoral body. The ischiadic tuberosities, the femoral head, the lesser trochanter and other circular like structures are extracted using variants of the Hough transform for circles.

  The other parameters that are important in hip arthroplasty are computed based on the previously mentioned ones. Even if the automatic extraction method is not 100% accurate, it improves the medical process by reducing the time spent for the detection of parameters before and after the surgical intervention. It also gives the possibility to manually adjust the wrongly extracted parameters and to perform other manual measurements.

- The Canny edge detector and the Hough transforms are time consuming. This chapter also describes our contribution to the reduction of computing times for these feature extraction algorithms.

  The first solution reduces the search spaces in the Hough transforms based on the particularities of the investigated radiographic images and on the searched parameters.

  The other approaches are based on the GPGPU technology, with the focus on the possibility to use multi-GPGPU accelerated computers and other mechanisms of the CUDA architecture. The comparison between the computing times of the single- and multi-GPGPU implementations of the Canny edge detector and the Hough transform for lines demonstrates the scalability of the multi-GPGPU approach. We also reduced the overhead of the memory transfers between the CPU and the GPU by using the page locked memory.

This chapter shows a comparison between the CPU and GPU implementations of the feature extraction algorithms and presents the advantages and disadvantages of each new method.

The proposed techniques can be used for feature extraction in any kind of images containing salient formations that are approximated by lines or circles. The idea is to first observe the particularities of the investigated images, which can reduce the search space for the parameters of the Hough transforms.

Also, this chapter proves that the parallel implementations that use the power of the graphic cards add an impressive performance gain to the object extraction algorithms.

CHAPTER 3

RECONSTRUCTION AND VISUALIZATION OF MEDICAL DATA

The theoretical and practical contributions described in this chapter have been (or will be) published in the following papers:

L. Petrescu, A. Morar, F. Moldoveanu, V. Asavei, "Real Time Reconstruction of Volumes from Very Large Datasets using CUDA", Proceedings of the 15th International Conference on System Theory, Control and Computing, pp. 462-466, Sinaia, 2011.

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, C. Boiangiu, A. Egner, "Computer Assisted Analysis of 2D/3D Medical Images", Proceedings of the 21st International DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 1273-1274, Zadar, 2010.

A. Morar, F. Moldoveanu, V. Asavei, L. Petrescu, A. Moldoveanu, A. Egner, "GPGPU Based Non-photorealistic Rendering of Volume Data", accepted for publication in the Journal of Control Engineering and Applied Informatics (to appear in 2013).

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "Computer Assisted Insertion of Prostheses Based on Medical Images", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.636-641, Bucharest, 2011.

This chapter presents our contributions in the field of volume rendering. The first contribution is a memory efficient implementation of the marching cubes algorithm using Nvidia's CUDA technology. The algorithm can handle datasets that are normally too large for current hardware. The second contribution consists of three methods of non-photorealistic rendering that extract silhouettes from relatively large datasets.

3.1. REAL TIME RECONSTRUCTION OF VOLUMES FROM LARGE DATASETS

Iso-surface extraction is a common technique in volume visualization. Iso-surfaces are surfaces that contain points which have the same scalar value within a volume dataset. Usually, for a CT dataset, this value represents a radiodensity of interest, measured in Hounsfield Units (HU). For instance, in CT images, the radiodensity of bone tissue ranges between 700 and 3000 HU, while the radiodensity of soft tissue is between -300 and -100 HU.

It is often useful to have the mesh representation of these iso-surfaces available for computation of properties, such as surface area, or for processes that require the geometric representation of the surface. As previously stated, we have worked on a project whose aim was to automatically or semi-automatically generate 3D models of custom prostheses for hip arthroplasty. For this procedure, the geometric representation of the femoral bone surface is required.

Usually, the user has a clear idea of what structures are of interest in a dataset. However, the user might not know exactly what iso-value should be used to extract these structures. Therefore a fast iso-value changing mechanism is vital for real time applications that require surface reconstruction.

The most widely used technique for surface reconstruction is the marching cubes algorithm, which is described in section 1.4.2.

The main problem of 3D medical image processing is the large size of the data acquired from medical devices. One CT image usually has 512^2 pixels, and a CT dataset can sometimes contain even 2048 slices. Another challenge of medical applications is that they are generally required to work in real time. The increased complexity of the marching cubes algorithm, caused by the huge number of intersections that are processed, implies high memory usage and slow computing times. The time problem can be solved with the parallel implementation of marching cubes that is described in [29]. Table 3.1 shows a comparison between the computing times of the marching cubes algorithm, executed on the CPU and on the GPU with CUDA, for the same datasets, with the same iso-value. The tests were made on an i7-720QM processor 1.60 GHz with 4GB RAM and an Nvidia GeForce GT 230M GPU with 1GB GPU RAM. The CUDA implementation runs approximately 300 times faster than the CPU one, for datasets of size 256^3. However, the original implementation of the marching cubes algorithm with CUDA can handle datasets of up to 512^2x64 with 1.2 GB of GPU RAM required. This thesis proposes a new algorithm that gives the possibility to reconstruct surfaces from much larger datasets without using surface generalization or simplification.

Table 3.1. Computing times for marching cubes on the CPU and on the GPU

Dataset size, number of triangles generated	Computing times for marching cubes on the CPU (ms)	Computing times for marching cubes with CUDA (ms)	Performance gain
256^2x128	12870	53.98	238x
256^2x128	12637	53.14	237x
256^3	11616	38.22	303x
256^3	23054	74.59	309x
256^3	29594	87.22	339x

3.1.1. Classic Marching Cubes with CUDA

This section shortly describes the pseudo code of the marching cubes algorithm with CUDA from Nvidia's CUDA SDK Example [29], with a small change. In the classic implementation, the normals of the triangles that approximate the iso-surface are computed based on the triangle vertices using the cross product. Thus, only one normal is generated for each triangle. In our method, the triangle normals are replaced by gradient vectors. The gradient vectors of the voxels' vertices are approximated with finite differences based on the scalar values within the volume. The gradient vectors of the iso-surface triangles are computed using linear interpolation from the gradient vectors of the voxels' vertices. This change offers a better visual effect to the rendered iso-surface by allowing smooth shading. In this chapter gradient vectors and normal vectors are used interchangeably.

The vertices and normal vectors are stored into vertex buffer objects (VBOs) in order to reside in the GPU memory rather than the system memory. This way we obtain a substantial performance gain over passing the data between CPU and GPU due to the fact that the data can be rendered directly by the GPU.

Pseudo code 3.1. Marching Cubes with CUDA

```
Step 1. Copy data to the GPU
copy the search tables and the volume vol into texture memory on the GPU
Step 2. Classify and remove non-border voxels
for every voxel v in vol do (in parallel using a CUDA kernel)
        classify v (interior/exterior/border voxel)
end for
read total number of active (border) voxels
for every voxel v in vol do (in parallel using a CUDA kernel)
        if v is border voxel then
                save v in a vector of active voxels
        end if
end for
```

```
Step 3. Compute triangles of the iso-surface
for every active voxel av do (in parallel using a CUDA kernel)
        for every vertex vx of av do
                read value of vx and value of neighboring vertices (for the approximation of the
                                                                gradient vector with finite differences)
                compute gradient at vx
        end for
        for every intersection point ip of the iso-surface with the edges of av do (determined based on
                                                                the edge table)
                determine value of ip and normal at ip (using linear interpolation)
        end for
        for every triangle t of the iso-surface do (determined based on the triangle table)
                save vertices and normals into VBOs
        end for
end for
End of pseudo code
```

3.1.2. A Novel Approach to Marching Cubes with CUDA

This section describes the ideas behind a new parallel marching cubes algorithm that can handle relatively large datasets.

The proposed method divides the data volume into sub-volumes, in order to be processed serially on the GPU. Initially, the GPU stores only the lookup tables, i.e., the edge and the triangle table. After that, the initial volume is partitioned into chunks, as intuitively explained in Fig. 3.1.

The initial volume is divided into sub-volumes on the maximum size axis, which usually is the *z* axis (height). For example, if we have a CT dataset of 2048 tomograms, each of 512^2 pixels, the volume is divided into 64 chunks, each with 32 slices of 512^2 pixels. The GPGPU marching cubes is then applied on each of these sub-volumes, serially.

The main advantage of this method is that it requires much less memory than the original CUDA marching cubes, without division of the data volume. Also, for sparse volumes, empty chunks are culled in one operation. If a chunk contains only empty voxels, it will be labeled as an irrelevant chunk at loading time. This is why the chunk will not be further processed with voxel labeling or marching cubes. Therefore the speed and memory requirements of the new method greatly outmatch those of the original CUDA marching cubes algorithm. Another advantage is that the division of the volume provides the possibility to reconstruct only parts of the volume. Thus, if a chunk is not of interest, it doesn't require reconstruction.

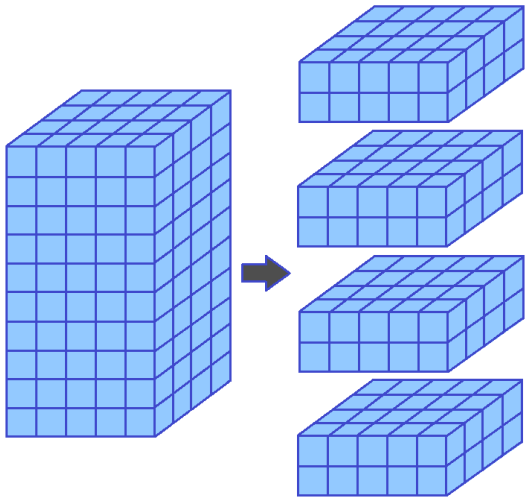

Fig. 3.1. Splitting the initial volume (left) into chunks (right)

The size of the chunks can be easily changed. However, the chunk overlapping could become an issue. The volume is composed of slices that represent sections through the scanned body. The volume voxels contain vertices from two adjacent slices, four on one slice and four on the other one. In order to reconstruct the surface correctly with no free spaces between the sub-surfaces reconstructed from adjacent chunks, the border slices must be duplicated. As can be observed in Fig. 3.2, two adjacent chunks share a common slice, which is the first slice for a chunk and the last one for the other chunk. Therefore the size of the chunks has to be chosen in such a manner that overlapping would not lead to very high memory usage. For example, if the chunks are composed of only two slices, the whole slices in the dataset are duplicated.

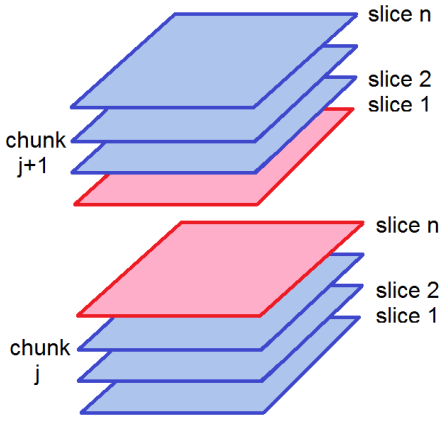

Fig. 3.2. Chunks overlapping. The first slice in chunk j+1 (slice 1) is the same as the last slice in chunk j (slice n)

Furthermore the algorithm can be specifically tailored to use different chunk sizes for different parts of the volume if there is the knowledge that some areas are much more detailed than others.

The only disadvantage of the proposed method is that it is slightly slower than the original CUDA marching cubes, because the chunks are computed serially. Even though in the original algorithm the CUDA driver also serializes the process due to lack of multiprocessors, our serialization swaps volume data between CPU and GPU. Nevertheless the advantages greatly exceed this drawback.

The pseudo code of this new algorithm is divided into two stages:

- the preprocessing stage, executed before the modified surface reconstruction algorithm
- the reconstruction stage

Pseudo code 3.2. Modified Marching Cubes with CUDA

```
Preprocessing stage
create textures to store the edge table and the triangle table on the GPU
determine the maximum size of a sub-volume based on the size of the GPU RAM (usually size =8/16/32)
for each sub-volume sv do (in parallel using a CUDA kernel)
        scan every voxel v
        if all the voxels of sv are not of interest, then
                mark sv as unimportant
        end if
end for
Reconstruction stage
for every sub-volume sv do
        if sv is of interest, then
                scan sv and remove non-active voxels  (similar to Step 2 from Pseudo code 3.1)
                compute triangles of the iso-surface (similar to Step 3 from Pseudo code 3.1)
        end if
end for
End of pseudo code
```

3.1.3. Results

The implementation of the presented algorithm has been tested on a GTX 460M GPU with 1.5 GB GPU RAM. Various sizes have been chosen for the chunks, in order to compare the memory usage and draw some conclusions regarding the best choice for the chunk size. Table 3.2 presents the results of the runs for every size.

Table 3.2. Memory usage for different chunk sizes

Chunk size (number of slices)	RAM (MB)	Memory usage (%)
32	1495	99.6
16	1440	96
8	1432	95.4
4	>1500	>100

From these results we can draw the conclusion that chunks of size 32 offer the best performance/overlapping ratio, thus being chosen for further runs. Furthermore it can be concluded that overlapping can become a major memory problem if the chunk size is too small.

For a dataset with 15 million vertices that need to be rendered, the application runs at 2.3 fps if the chunks are being reconstructed each frame. But, for faster results, it is better to process the chunks only when reconstruction is necessary, namely each time the iso-value for the iso-surface is changed by the user. This way, for the same number of vertices, the application runs at 43.4 fps (which is more than 24 frames per second - the threshold for real-time applications).

Fig. 3.3 presents the results for reconstructing a 3D surface based on a CT dataset of size $1024x512^2$ representing sections through a human body.

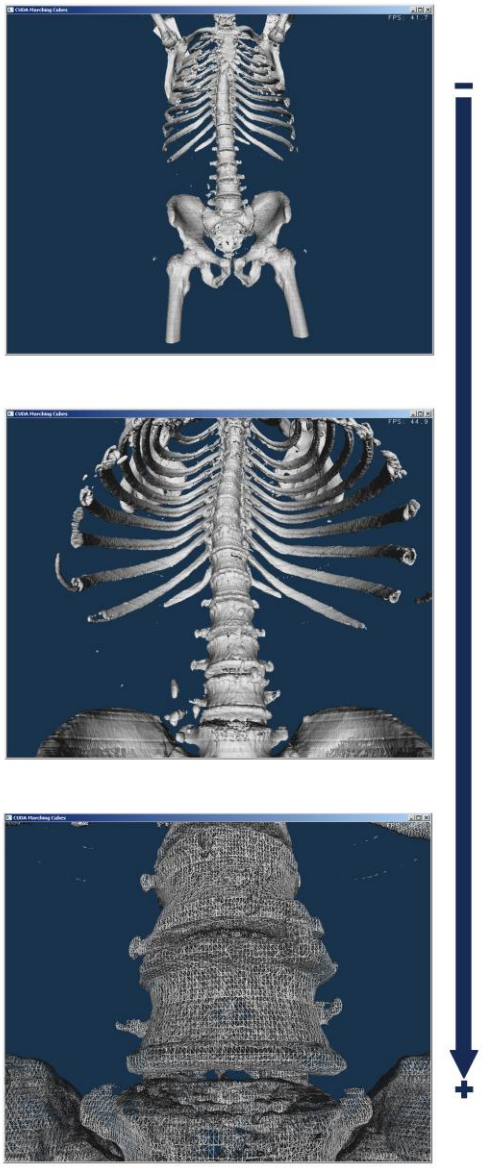

Fig. 3.3. 3D reconstructed surface at different levels of detail. The third image is rendered wireframe in order to show how detailed/large the dataset is.

The iso-value has been chosen so that the rendered iso-surface would represent only bone tissue. The screenshots illustrate the reconstructed surface at different levels of detail, in order to observe that this application is relatively noise free. The accuracy of the 3D surface is limited only by the accuracy and correctness of the dataset, even though in our implementation we use filtering operations on the dataset to remove noise and improve the overall quality of the data volume that will be used by the algorithm.

Compared to the algorithm presented in [37] which is focused on handling datasets up to 77 GB but does not reconstruct the iso-surface in real time, the aim of our algorithm is to reconstruct the iso-surface in real time, using smaller datasets (26 GB). In the SABIMAS project [39], the application allows the medical experts to examine the 3D model of the femur bone by viewing it from different positions (making rotations) and at different levels of detail.

3.2. GPGPU BASED NON-PHOTOREALISTIC RENDERING OF VOLUME DATA

This section describes three methods of extracting silhouettes from relatively large datasets very fast (in some cases, even in real time), using the GPGPU technology. These methods are suitable for different types of datasets, applications and hardware characteristics. The first method extracts the iso-surface and then computes its silhouette. The second one extracts only the silhouette and computes the visibility of the contour vertices using an algorithm inspired from ray casting. The third method uses a CUDA rasterizer in order to render iso-surfaces and silhouettes from large datasets.

All the silhouette rendering algorithms presented in this chapter follow a parallel implementation. A CUDA kernel that is defined for one data element is run in parallel for all the data elements. The first silhouette rendering technique is very fast, but uses a lot of GPU memory. This is the reason we designed other two algorithms that require less GPU memory. All the three methods are based to some extent on the marching cubes CUDA implementation from the CUDA SDK Example [29] and the non-photorealistic rendering method proposed by Burns et al. [27]. The GPGPU marching cubes algorithm is slightly detailed in section 3.1.1. The silhouette extraction method of Burns et al. is discussed in section 1.4.3.

The change to the marching cubes algorithm described in 3.1.1 is also useful for the silhouette rendering methods. The silhouette extraction presented in 1.4.3 is based on the difference between the normal vectors of a triangle's vertices. Thus, the original algorithm with a single normal for the entire triangle cannot be used.

Changing the viewer's position does not affect the performance of the marching cubes rendering method. This is why our first approach is based mainly on the implementation of marching cubes with CUDA.

In order to handle relatively large datasets, the original volume can be divided into smaller sub-volumes that are reconstructed serially on the GPU, as discussed in section 3.1.2. In the original GPGPU marching cubes algorithm, the results of the intersection between the iso-surface and the volume voxels, i.e., the triangle vertices and their normals, are stored into two

VBOS. In the algorithm described in 3.1.1, instead of having two VBOs for the whole volume, two VBOs are used for each sub-volume. This implementation is slightly slower than the original CUDA implementation, but can handle considerably larger datasets.

3.2.1. Silhouette Rendering with Marching Cubes and Geometry Shader

This new approach takes advantage of the flexibility of the programmable rendering pipeline. After the extraction of the triangular mesh that approximates the iso-surface, the silhouette is rendered using a vertex shader and a geometry shader.

The input of the vertex shader consists of the position and normal vector in object coordinates for each vertex, the eye position in world coordinates, the model matrix and the model-view-projection matrix. The output of the vertex shader contains the vertex position in clip space, the position and normal vector and the eye position in world coordinates.

The following code represents the vertex shader program.

Code 3.1. Vertex shader program

```
void silhouette_vp(
        float4 position                 : POSITION,     //position in object coordinates
        float4 color                    : COLOR,
        float3 normal                   : NORMAL,       //normal in object coordinates
        float3 eye,                                     //eye in world coordinates
        out float4 o_position           : POSITION,     //position in clip space
        out float4 o_color              :COLOR,
        out float4 o_position_world     : TEXCOORD0,    //position in world coordinates
        out float3 o_normal_world       : TEXCOORD1,    //normal in world coordinates
        out float3 o_eye_world          : TEXCOORD2,    //eye in world coordinates
        uniform float4x4 model ,                        //model matrix
        uniform float4x4 modelViewProj)                 //model view projection matrix
{
  o_position = mul(modelViewProj, position); //obtain the position of the vertex in clip space
  o_position_world = mul(model,position); //compute the position of the vertex in world coordinates
  o_normal_world = normalize(mul((float3x3)model,normal)); //the normal vector in world coordinates
  o_eye_world = eye;  //eye position in world coordinates
  o_color=color; //color
}
```

While the vertex shader can process only vertices, the geometry shader can handle whole primitives. The input of the geometry shader is represented by a mesh triangle in world and clip coordinates and the eye position in world coordinates. The output contains either nothing, or a segment of the silhouette iso-surface in clip coordinates, computed based on the silhouette extraction algorithm exemplified in Fig. 1.15. For every triangle of the iso-surface that intersects

the surface where the normal is perpendicular to the view, the geometry shader emits a silhouette segment representing that intersection.

The following code contains the geometry shader program.

Code 3.2. Geometry shader program

```
// output lines that are part of the silhouette based on the characteristics of the input triangles
TRIANGLE LINE_OUT
void silhouette_gp(
          AttribArray<float4> pos              : POSITION,   //the pos of the triangle vertices in clip space
          AttribArray<float4> pos_world    : TEXCOORD0, //the positions in world coordinates
          AttribArray<float3> norm_world  : TEXCOORD1, //the normal vectors in world coordinates
          float3 eye_world                      : TEXCOORD2, //the eye position in world coordinates
          float4 color                             :COLOR)
{
   float t1,t2;
   float4 pos1,pos2;

  //For each triangle vertex, determine the vector from the eye to the vertex
   float3 view[3];
   view[0]=eye_world-pos_world[0].xyz; //view vector for the first triangle vertex
   view[1]=eye_world-pos_world[1].xyz; //view vector for the second triangle vertex
   view[2]=eye_world-pos_world[2].xyz; //view vector for the third triangle vertex

   //Compute the dot product between the view vector and the normal vector for each vertex
   float dot_vn[3];
   dot_vn[0]=dot(view[0],norm_world[0]); //dot product for the first triangle vertex
   dot_vn[1]=dot(view[1],norm_world[1]); //dot product for the second triangle vertex
   dot_vn[2]=dot(view[2],norm_world[2]); //dot product for the third triangle vertex

   //Compute a flag that contains information about the value of each dot product relative to 0
   float tri_index=0;
   tri_index+=(dot_vn[0]<0);     //if the first dot product is negative
   tri_index+=(dot_vn[1]<0)*2; //if the second dot product is negative
   tri_index+=(dot_vn[2]<0)*4; //if the third dot product is negative

   //if not all dot products are positive/negative, we have a silhouette segment
   if ((tri_index!=0)&&(tri_index!=7))
   {
      if (tri_index==1 || tri_index== 6) //the first dot product has a different sign than the other two
      {
         t1=(0-dot_vn[0])/(dot_vn[1]-dot_vn[0]);
         t2=(0-dot_vn[0])/(dot_vn[2]-dot_vn[0]);
         pos1=lerp(pos[0],pos[1],t1);  //the position of the first silhouette point
```

```
        pos2=lerp(pos[0],pos[2],t2);  //the position of the second silhouette point
        //Emit the silhouette line
        emitVertex( pos1: POSITION, color : COLOR0);
        emitVertex( pos2: POSITION, color : COLOR0);
      }
      else
        if (tri_index==2 || tri_index==5) //the second dot product has a different sign than the other two
        {
          t1=(0-dot_vn[1])/(dot_vn[0]-dot_vn[1]);
          t2=(0-dot_vn[1])/(dot_vn[2]-dot_vn[1]);
         pos1=lerp(pos[1],pos[0],t1);
         pos2=lerp(pos[1],pos[2],t2);
        //Emit the silhouette line
         emitVertex( pos1: POSITION, color : COLOR0);
         emitVertex( pos2: POSITION, color : COLOR0);
        }
        else  //the third dot product has a different sign than the other two
        {
          t1=(0-dot_vn[2])/(dot_vn[0]-dot_vn[2]);
          t2=(0-dot_vn[2])/(dot_vn[1]-dot_vn[2]);
          pos1=lerp(pos[2],pos[0],t1);
          pos2=lerp(pos[2],pos[1],t2);
        //Emit the silhouette line
          emitVertex( pos1: POSITION, color : COLOR0);
          emitVertex( pos2: POSITION, color : COLOR0);
        }
      }
}
//End of code
```

In order to compute the visibility of the silhouette points, the geometry representing the iso-surface is sent to the graphics pipeline, and rasterized only for the creation of the depth buffer. After this step, the contour lines are also rasterized by updating both the color and the depth buffer. Thus, only the silhouette pixels that are not occluded are rendered. Fig. 3.4(a) shows the iso-surface reconstructed with the marching cubes algorithm. The vertices and normal vectors of the iso-surface are sent to the vertex shader and then to the geometry shader. Fig. 3.4(b) illustrates the output of the geometry shader which is represented by the silhouette rendered without any visibility test. The output of the algorithm, i.e., the silhouette rendered with the visibility test, is shown in Fig. 3.4(c).

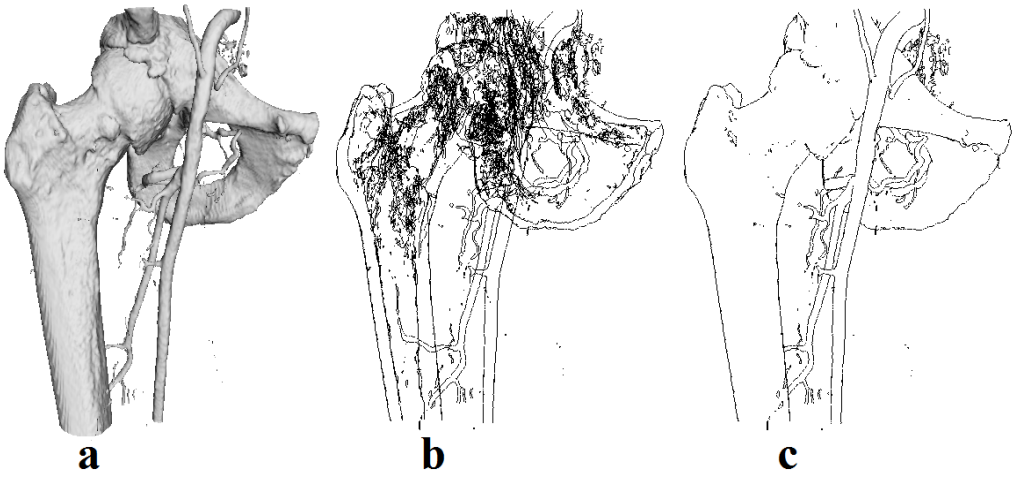

Fig. 3.4. Volume rendering: (a) iso-surface rendered with marching cubes (b) silhouette rendered without visibility test, (c) silhouette rendered with visibility test

The silhouette rendering flow described in this section is depicted in Fig. 3.5.

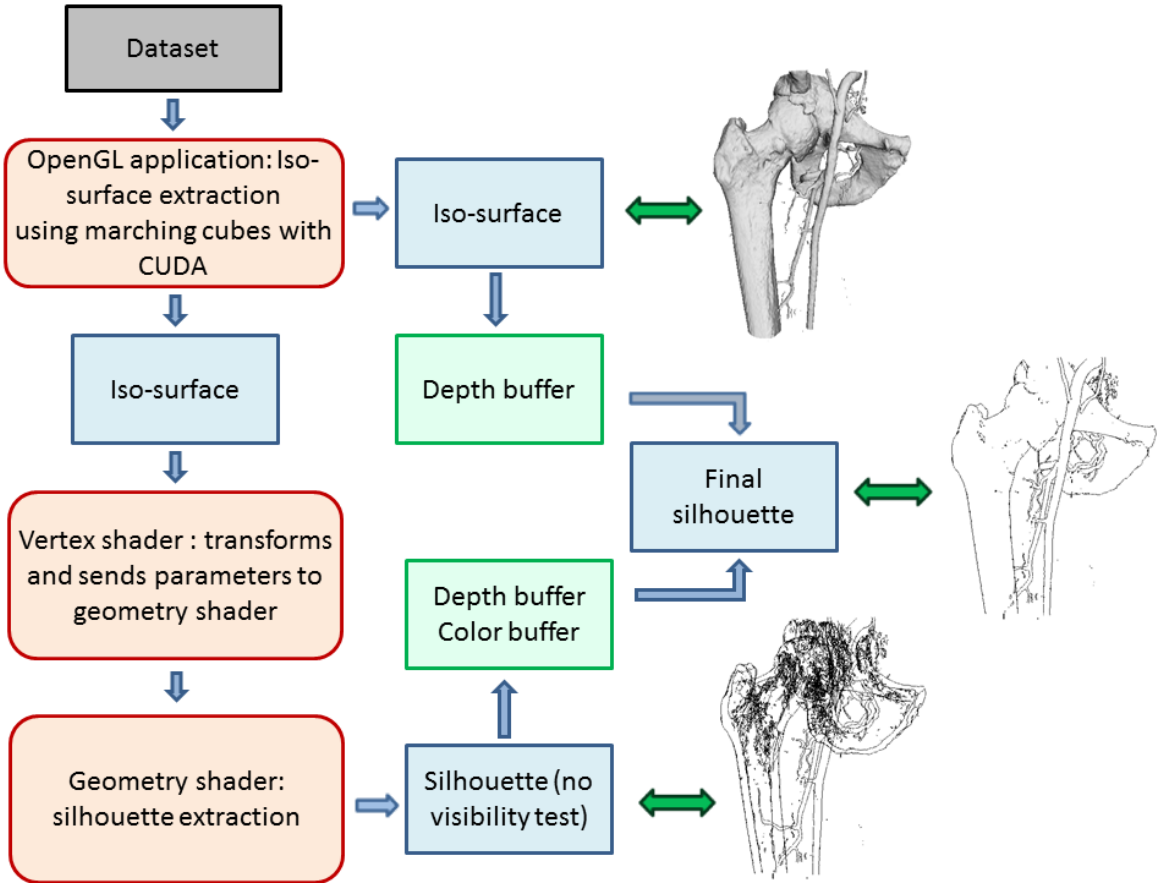

Fig. 3.5. The flow of the algorithm that renders the silhouette with marching cubes and geometry shader

This method is very fast, but requires a lot of GPU memory space. If both the volume and the geometry are stored on the GPU, then this implementation is suitable only for small datasets (up to 256^3). Still, it can process larger datasets, but with an improvement.

We can use an approach similar to the one described in section 3.1 in order to handle larger datasets. The volume is divided into sub-volumes that are processed serially on the GPU. The geometry which consists of two VBOs for each sub-volume is further processed by the vertex and geometry shader in the same manner as the one described for small datasets. This approach solves to some extent the memory limitations, because, at one moment in time, only a small part of the volume is computed. But the need to store the whole geometry on the graphics card still limits the size of the datasets that can be handled.

3.2.2. Silhouette Rendering with Ray Based Visibility Test

The second approach does not save the geometry on the GPU, but renders only the visible points that belong to the silhouette. The visibility test is based on the method of Burns et al. [27].

First, the silhouette is extracted without any visibility test. Then, for every vertex belonging to the silhouette, a ray is cast through the volume from the eye to that vertex, intersecting it with the iso-surface. If the ray goes from inside to outside or from outside to inside the iso-surface, then the current silhouette point is occluded. The steps of the algorithm are described in Fig. 3.6.

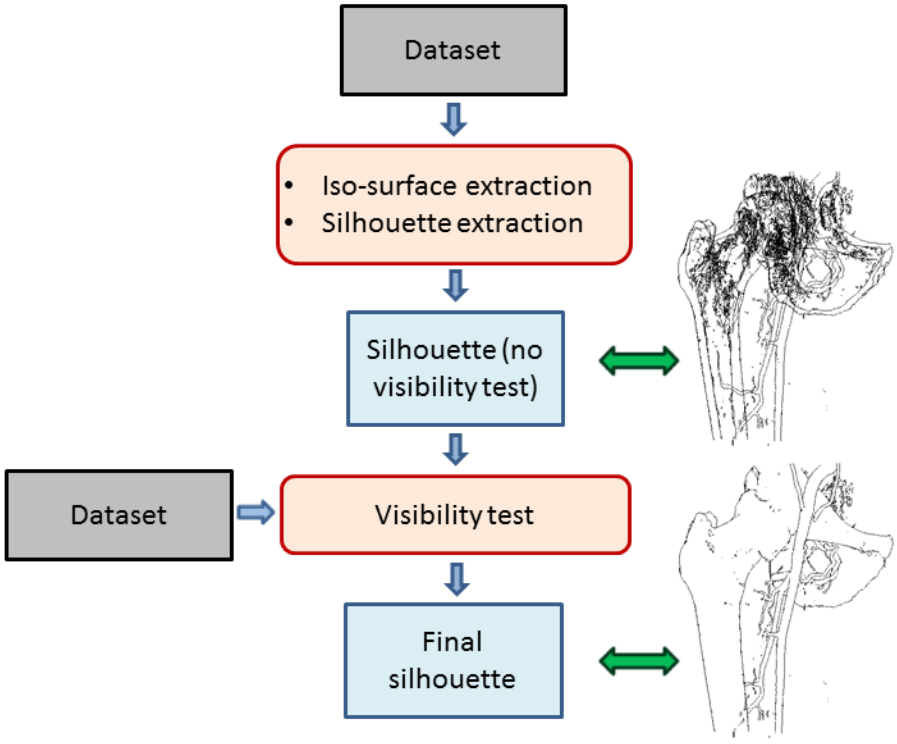

Fig. 3.6. The flow of the silhouette rendering algorithm with ray based visibility test

The visibility test is based on the implementation of the ray-casting algorithm ("volumeRendering" application) from the CUDA SDK Example [29]. The algorithm for silhouette rendering with ray based visibility test is described in the following pseudo code:

Pseudo code 3.3. Silhouette rendering with ray based visibility test

```
Step 1. Similar to Step 1 from pseudo code 3.1
Step 2. Similar to Step 2 from pseudo code 3.1
Step 3. Extract silhouette lines that are visible
for every active voxel av from volume vol do (in parallel using a CUDA kernel)
        determine the triangles that approximate the iso-surface (similar to Step 3 from 3.1)
        for every triangle t that approximates the iso-surface, do
                for every vertex vx do
                        compute the view vector vv
                        compute the dot product dp between normal at vx and vv
                end for
                if dp for one vertex has a different sign from the dp-s for the other two vertices, then
                        compute intersection of t with the surface where the normal is perp. to the view
                        (based on the algorithm from section 1.4.3; the result is a silhouette segment s)
                        for each endpoint p of s do
                                cast a ray r from eye to p
                                determine where r enters vol
                                for every ray step, do (the step size is equal to the size of a voxel edge)
                                        determine the next intersected voxel iv
                                        (by incremental computing)
                                        (iv is the voxel that contains the next point on the ray)
                                        compute the intersection of iv with r
                                        determine the face f of iv by which r leaves iv
                                        determine the value of f
                                        (with linear interpolation from f's corners)
                                        if the value of f has changed sign, then
                                                p is occluded
                                        end if
                                end for
                        end for
                        if both endpoints of s are visible, then
                                store s into a VBO
                        end if
                end if
        end for
end for
End of pseudo code
```

The algorithm should work for larger datasets because only the silhouette is stored on the GPU, while in the first approach the whole iso-surface is stored on the graphics card. However, the performance is altered by the viewer's position. Each time the eye position is changed, the silhouette has to be computed all over again. Also, the visibility test is time consuming because of the large amount of computations for the rays cast from the eye position to the silhouette vertices.

For larger datasets the initial volume can be divided into chunks. The processing flow is explained in Fig. 3.7 in order to provide some insight into the silhouette rendering with ray based visibility test for large datasets.

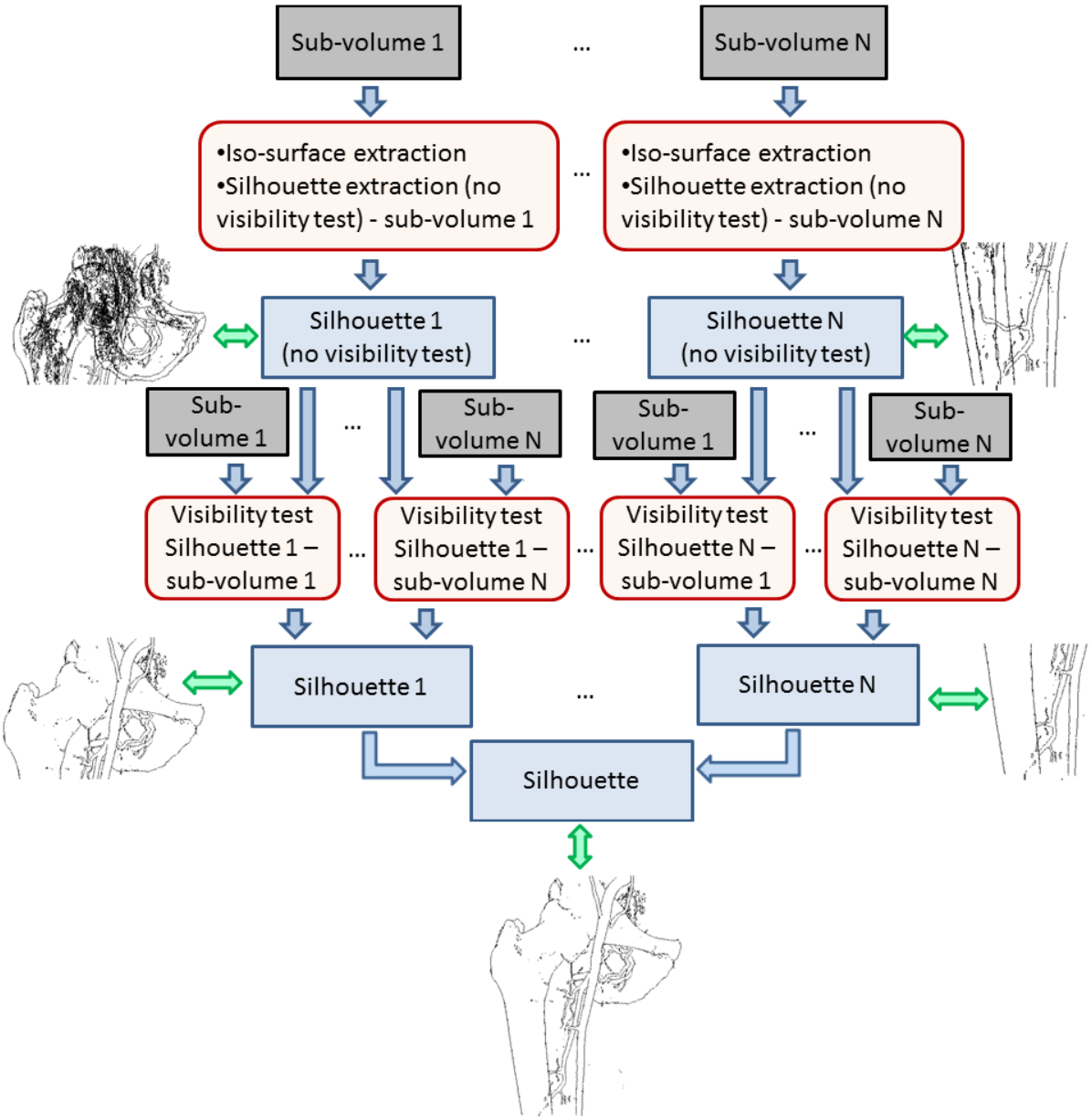

Fig. 3.7. The flow of the algorithm that renders silhouettes using a ray based visibility test – the approach that divides the volume into sub-volumes

This approach does not allow the generation of visible silhouette lines using a single CUDA kernel. Step3 from pseudo code 3.3 is divided into two steps. The first step, the generation of silhouette segments, both visible and occluded, is defined in one CUDA kernel. The kernel is run for all the sub-volumes. The generated silhouettes are referred to as sub-silhouettes. The process is serialized, meaning that at one moment in time only one sub-volume is processed in parallel on the GPU. The second step, i.e., the visibility test, is defined in another CUDA kernel. This kernel is run for all the sub-silhouettes, for every silhouette segment. The difference between the previous approach is that this time the visibility of a sub-silhouette is run not only for the corresponding sub-volume, but for all the sub-volumes, serially.

3.2.3. Silhouette rendering using a CUDA rasterizer

This section proposes a new method that solves the GPU memory limitation. It does not save the geometry, but renders it directly using a CUDA rasterizer. The rasterizer follows the implementation of the z-buffer algorithm. We use three buffers with the same size as the display window size. The first depth buffer stores the maximum depth values for the triangles that approximate the iso-surface. The second depth buffer stores the maximum depth values for the silhouette lines. The color buffer is updated based on the values of the depth buffers. The flow of the algorithm is illustrated in Fig. 3.8.

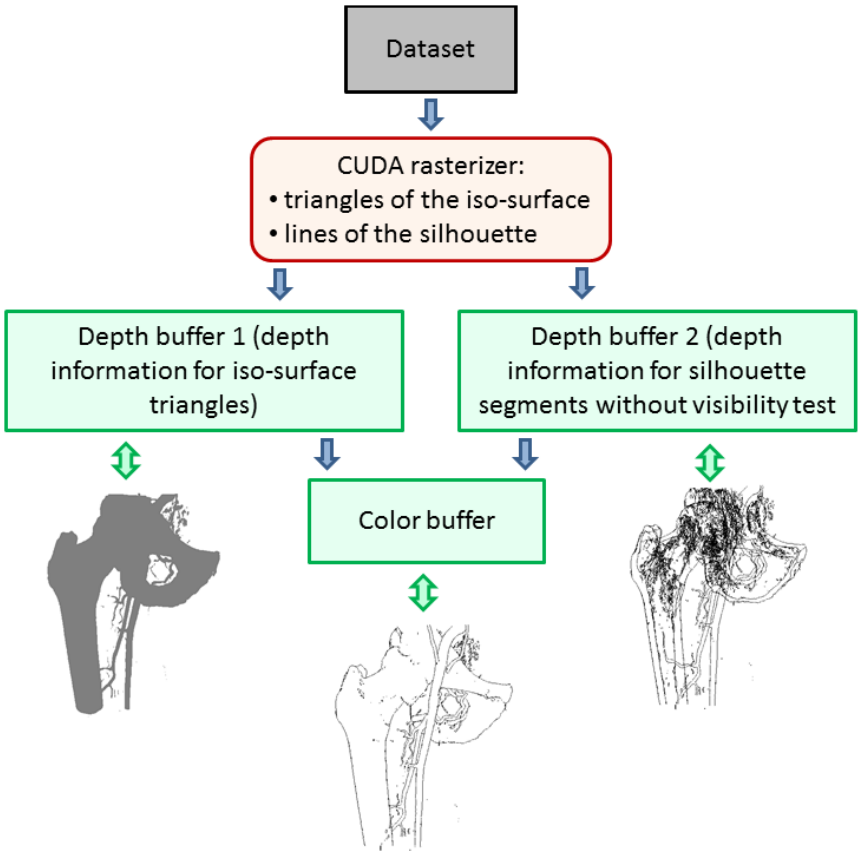

Fig. 3.8. The flow of the algorithm that renders silhouettes with a CUDA rasterizer

The pseudo code of the new algorithm is presented below.

Pseudo code 3.4: Silhouette rendering with CUDA rasterizer

```
Step 1.Copy volume to the GPU
save volume vol on GPU in texture memory
Step 2. Update depth buffers
for every voxel v in vol do (in parallel using a CUDA kernel)
        determine the triangles that approximate the intersection of v with the iso-surface
        for every triangle t, do (t is computed in normalized device coordinates)
                for every point (x,y,z) inside t or belonging to its edges, do
                        if zbuffer1[x,y]<z then
                                zbuffer1[x,y]=z
                        end if
                end for
                if the triangle contains a silhouette line s, then
                        for every point (x,y,z) belonging to s, do
                                if zbuffer2[x,y]<z then
                                        zbuffer2[x,y]=z
                                end if
                        end for
                end if
        end for
end for
Step 3. Update color buffer
for every pixel (x,y) on the display window do (in parallel using a CUDA kernel)
        cbuffer[x,y]=0
        if zbuffer2[x,y]>=zbuffer1[x,y] then
                cbuffer[x,y]=1
        end if
end for
End of pseudo code
```

The depth buffers are updated using the CUDA atomic function `atomicMax()` in order to avoid read-modify-write hazards. The color buffer is stored into a pixel buffer object (PBO) on the graphics card. This way the color buffer is rendered directly by the GPU.

This implementation is slower than the first approach, because the proposed CUDA rasterizer is not specialized for rendering like the hardware rendering mechanisms of the GPU. Also, the generation of visible silhouette segments has to be repeated every time the eye position is changed.

The CUDA rasterizer can be used also for indirect volume rendering. If there is not enough GPU memory to store the iso-surface with marching cubes, the geometry can be rasterized directly with this new approach.

For larger datasets, the same depth buffer updating kernel (Step 2 from pseudo code 3.4) can be run for each sub-volume. The color buffer updating kernel (Step 3) is run only once, just like in the previous pseudo code.

3.2.4. Results

Fig. 3.9 shows a silhouette rendered from a volume dataset of size 512^3 with our first method.

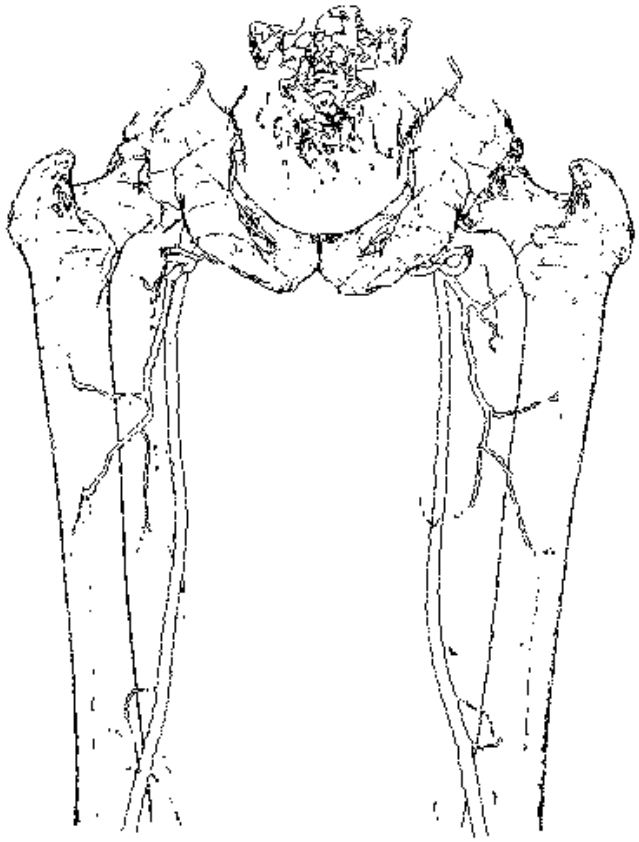

Fig. 3.9. Silhouette rendered with the method that uses marching cubes and geometry shader

Table 3.3 presents the computing times for the silhouette rendering methods described in the previous sections. The silhouettes belong to iso-surfaces from volume datasets of different sizes. The second column in the table refers to the reconstruction mode. In the first case, the reconstruction of the iso-surface/silhouette is done only when the iso-value is changed. This mode can be applied only for the first rendering method. In the second case, the reconstruction is done not only when the iso-value is changed, but also when the viewer's position is shifted. For each method there are two columns, one for the algorithm that handles small datasets and one for the algorithm that can handle larger datasets due to the volume splitting into chunks. The results

include also the rendering times, not only the times for computing and extracting the iso-surfaces and the silhouettes.

All the tests were made on an Nvidia GeForce GT 230M GPU with 1GB GPU RAM.

Table 3.3.Computing times for the three methods of silhouette rendering

Dataset size	Reconstr. mode	1st method without volume division (ms)	1st method with volume division (ms)	2nd method without volume division (ms)	2nd method with volume division (ms)	3rd method without volume division (ms)	3rd method with volume division (ms)
32^3	iso-val. changed	4.2	5.75	x	x	x	x
32^3	always	7.4	8.79	10.3	12.76	14.5	15.5
256^3	iso-val. changed	16	29.8	x	x	x	x
256^3	always	115	117	677	839.7	501	515
512^3	iso-val. changed	x	66.1	x	x	x	x
512^3	always	x	865	x	4030	x	2609
512^2x1024	iso-val. changed	x	x	x	x	x	x
512^2x1024	always	x	x	x	x	x	9872

As can be observed in Table 3.3, the first method is the fastest. If the reconstruction is done only when the iso-value is changed, the computing times are very small. However, this method can handle only small datasets, even with the volume division into sub-volumes, because the whole geometry is stored on the GPU. The second approach should handle larger datasets, but is limited by the fact that the silhouette is stored on the GPU. The third approach works faster than the second one for datasets of size 256^3 and 512^3. Theoretically, it can handle volume datasets of unlimited size, but at non-interactive frame rates.

Each implementation uses a different visibility test, with more or less influence on the computing times. In order to observe these influences, tests have been made for each method, with and without the visibility test.

Table 3.4 shows the computing times for the silhouette rendering technique that uses marching cubes and geometry shader. The second column has the same meaning as the one from Table 3.3. The third and fourth columns correspond to the method that does not divide the volume, and can handle only small datasets.

The difference between applying and not applying the visibility test in the first silhouette rendering method is almost insignificant, because the iso-surface is extracted in any case. The small increase in speed for the approach that does not test the visibility of the silhouette vertices is caused by the fact that the iso-surface is sent only once to the graphics pipeline. In the approach that tests the visibility of the silhouette vertices, the iso-surface is sent twice to the graphics pipeline, once for updating the depth buffer and once for the silhouette extraction.

Table 3.4. Computing times for the silhouette rendering method using marching cubes and geometry shader, with and without visibility test

Dataset size	Reconstruction mode	1st method without volume division - no visibility test (ms)	1st method without volume division - with visibility test (ms)	1st method with volume division - no visibility test (ms)	1st method with volume division - with visibility test (ms)
32^3	iso-val changed	4.2	4.2	5.7	5.75
32^3	always	7.4	7.4	8.6	8.79
256^3	iso-val changed	15	16	29.2	29.8
256^3	always	114.5	115	175	177
512^3	iso-val changed	x	x	63	66.1
512^3	always	x	x	815	865

Table 3.5 presents the computing times for the silhouette rendering algorithm with ray based visibility test.

Table 3.5. Computing times for the second silhouette rendering method, with and without ray based visibility test

Dataset size	2nd method without volume division - no visibility test (ms)	2nd method without volume division - with visibility test (ms)	2nd method with volume division - no visibility test (ms)	2nd method with volume division - with visibility test (ms)
32^3	6.7	10.3	10	12.76
256^3	121.5	677	143.4	839.7
512^3	x	x	782	4030

Here the differences between the two approaches are remarkable, especially for the method that handles large datasets by dividing the volume into sub-volumes. For the dataset of size 512^3, the algorithm that does not test the visibility of the silhouette vertices runs 5x faster than the one that performs the ray based visibility test. Thus it can be concluded that the ray based visibility test is very time consuming, especially for large volumes which have to be split into sub-volumes.

Table 3.6 shows the computing times for the silhouette rendering algorithm that uses a CUDA rasterizer.

Table 3.6. Computing times for the silhouette rendering method with a CUDA rasterizer

Dataset size	3rdmethod without volume division - no visibility test (ms)	3rdmethod without volume division - with visibility test (ms)	3rdmethod with volume division - no visibility test (ms)	3rdmethod with volume division - with visibility test (ms)
32^3	11.2	14.5	15	15.5
256^3	490	501	334	515
512^3	x	x	2213	2609
512^2x1024	x	x	5143	9872

The differences between applying the visibility test or not are also considerable, but small as compared to the differences from the second rendering method. In the implementation that tests the visibility of the silhouette vertices, the iso-surface is rendered with a CUDA rasterizer that is not specialized for rendering. The atomic operations that update the depth buffers represent a bottleneck for this approach. In the implementation that does not test the visibility of the silhouette vertices the depth buffer of the iso-surface does not have to be updated. Also, the depth buffer of the silhouette does not have to be updated with an `atomicMax()` instruction. In order to render the silhouette without the visibility test it is not necessary to know the z-value of the current pixel, but whether it is a silhouette pixel or not.

The visibility tests of the three methods lead to different visual results. The visibility test for the first and third algorithm is run for every point belonging to a silhouette segment. In the second implementation, the visibility test is run only for the endpoints of a silhouette segment. This is the reason the silhouette rendered with the second method has more discontinuities. Fig. 3.10 illustrates the visual differences between the proposed silhouette rendering algorithms.

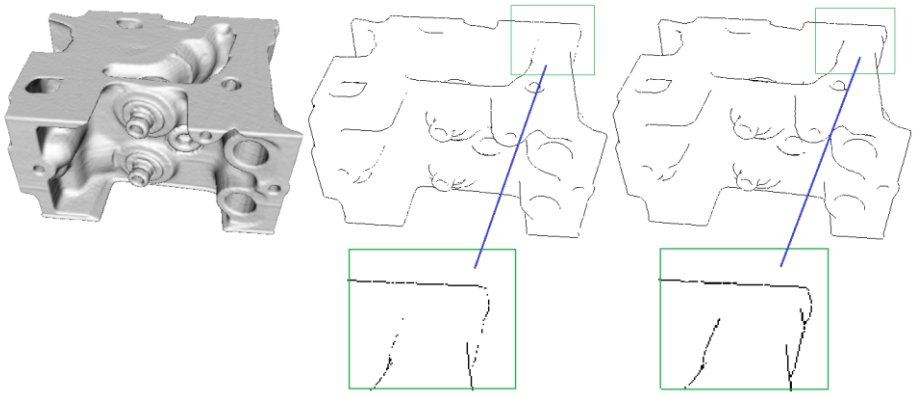

Fig. 3.10. Difference between rendering methods (from left to right): iso-surface rendering, silhouette rendering with ray based visibility test, silhouette rendering with geometry shader/CUDA rasterizer

3.3. CONCLUSIONS

- Our first contribution described in this chapter is a new implementation of the marching cubes algorithm, which can handle large datasets. The division of the volume into chunks has proven to offer excellent memory usage and to solve the problem of the GPU memory limitation in an optimal manner. The classic CUDA implementation of the algorithm works fast enough, though it cannot handle very large volumes on the current hardware. Instead, our algorithm is slightly slower, but can reconstruct surfaces for very large volumes ($2048x512^2$), up to 32 times larger than the original algorithm. If the reconstruction of the iso-surface is done only when the iso-value is changed, the application runs in real time.

  The results prove that this variant of the algorithm represents an improvement for real-time 3D surface reconstruction methods, especially in the medical field, where the size of the

processed data poses a problem. The application can be used by medical specialists to extract the 3D geometry of human body parts like bones. But, based on the obtained 3D model, other operations can be made. For example, the application should provide the possibility to alter the surface or make some automated measurements.

- This chapter also presents our contributions in the field of non-photorealistic volume rendering. We implemented three silhouette rendering methods that are suited for different types of datasets and different characteristics of the GPU cards.

  The first implementation requires enough GPU memory to store the dataset, the reconstructed iso-surface and the extracted silhouette. The advantages of this method are the fast rendering times and the possibility to reconstruct the iso-surface only when the iso-value is changed. The silhouette rendering with marching cubes and geometry shader leads to interactive frame rates for datasets of size up to 512^3 if the reconstruction is done only when the iso-value is changed.

  The second implementation stores on the GPU only the dataset and the silhouette of the iso-surface. The time consuming operation in the second method is the ray based visibility test, which casts rays from each silhouette vertex to the viewer's position, intersecting them with the volume. The implementation that splits the initial volume into chunks handles larger datasets, but is not very fast. The reason is that for one extracted sub-silhouette, the visibility test is run for each sub-volume, serially. Another drawback of this implementation is the requirement to extract the iso-surface and the silhouette each time the viewer's position changes.

  The third silhouette rendering implementation is perfectly suited for the case when the size of the dataset makes it impossible to store the iso-surface or the silhouette on the GPU memory. Our CUDA rasterizer solves the problem of the GPU memory limitation, but it is not as efficient as the hardware rendering mechanisms of the GPU. Similar to the second implementation, the iso-surface and the silhouette have to be extracted each time the position of the viewer is changed.

This chapter proves that the use of the GPGPU technology for volume visualization adds a considerably performance gain, as compared to the CPU approaches. It also shows that solutions exist for the problem of the large amount of data to be handled and the GPU memory limitation.

CHAPTER 4

PRODUCING PERSONALISED PROSTHESES FOR HIP ARTHROPLASTY STARTING FROM CT DATASETS

The theoretical and practical contributions described in this chapter have been published in the following papers:

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "Computer Assisted Insertion of Prostheses Based on Medical Images", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.636-641, Bucharest, 2011.

A. Morar, F. Moldoveanu, E. Gröller, "Image Segmentation Based on Active Contours without Edges", Proceedings of the 8th International Conference on Intelligent Computer Communication and Processing, Cluj, 2012.

A. Morar, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner, "CT Image Processing in Hip Arthroplasty", accepted for publication in The Scientific Bulletin of University POLITEHNICA of Bucharest, Series C Electrical Engineering and Computer Science (to appear in no.1, 2013).

As previously mentioned, we worked on a project whose aim was to automatically or semi-automatically generate 3D models of personalized prostheses for hip arthroplasty. In order to produce a prosthesis that fits very well the physiological characteristics of the femoral and pelvic bones of a patient, the geometric representation of the surface that approximates the bones is needed. For this, the marching cubes algorithm can be used (the algorithm was detailed in chapter 3). However, this algorithm cannot extract only one particular bone from the surface. This means that the femoral bone, the bone of interest for the exact fitting of the 3D model of the prosthesis, cannot be differentiated from the other bones in the human body. Also, because of the poor quality of the CT images, observed in the low contrast between bones and other tissues, the algorithm that extracts a surface based on a single iso-value might not produce the expected result.

The first section of this chapter proposes a flow for generating the 3D model of a prosthesis, based on the iso-surface obtained with marching cubes, representing the femur and the pelvic bone. The method has been partially implemented and tested, but it requires future improvements and testing.

An alternative to the proposed flow is to differentiate between different bones, to extract only the volume occupied by the femoral bone, and to further alter this volume for the purpose of generating the 3D model of the prosthesis. We designed two segmentation algorithms, one that extracts bones from individual CT slices and one that combines the results of the 2D segmentation in order to obtain the volume occupied by bones.

The second section of this chapter describes our 2D bone segmentation algorithm based on active contours without edges, which discriminates between bones and other tissues, but also between different bones.

The third section presents our 3D segmentation algorithm that extracts the volume occupied by different bones in the CT dataset.

4.1. SEMI-AUTOMATIC GENERATION OF 3D MODELS OF PROSTHESES FOR HIP ARTHROPLASTY

This section proposes some ideas for the development of a semi-automatic method to produce 3D models of artificial implants for hip arthroplasty. The method is based on the processing of both radiographic and CT images. In the first step of the method some parameters important in hip arthroplasty are extracted from a radiographic image. These parameters are used to adjust the 3D model of a prosthesis which can be produced using a modeling tool (such as Blender or 3D Max). Depending of the parameters' ranges, one or more 3D prosthesis models can be selected. Then, an initial estimate of the needed prosthesis is generated. The last stage in the process consists of fitting the 3D model of the generated prosthesis into the 3D model of the femoral bone obtained from a CT dataset. Fig. 4.1 illustrates the flow of the proposed method. The method has been partially implemented and requires further testing and improvements in order to be useful in practice.

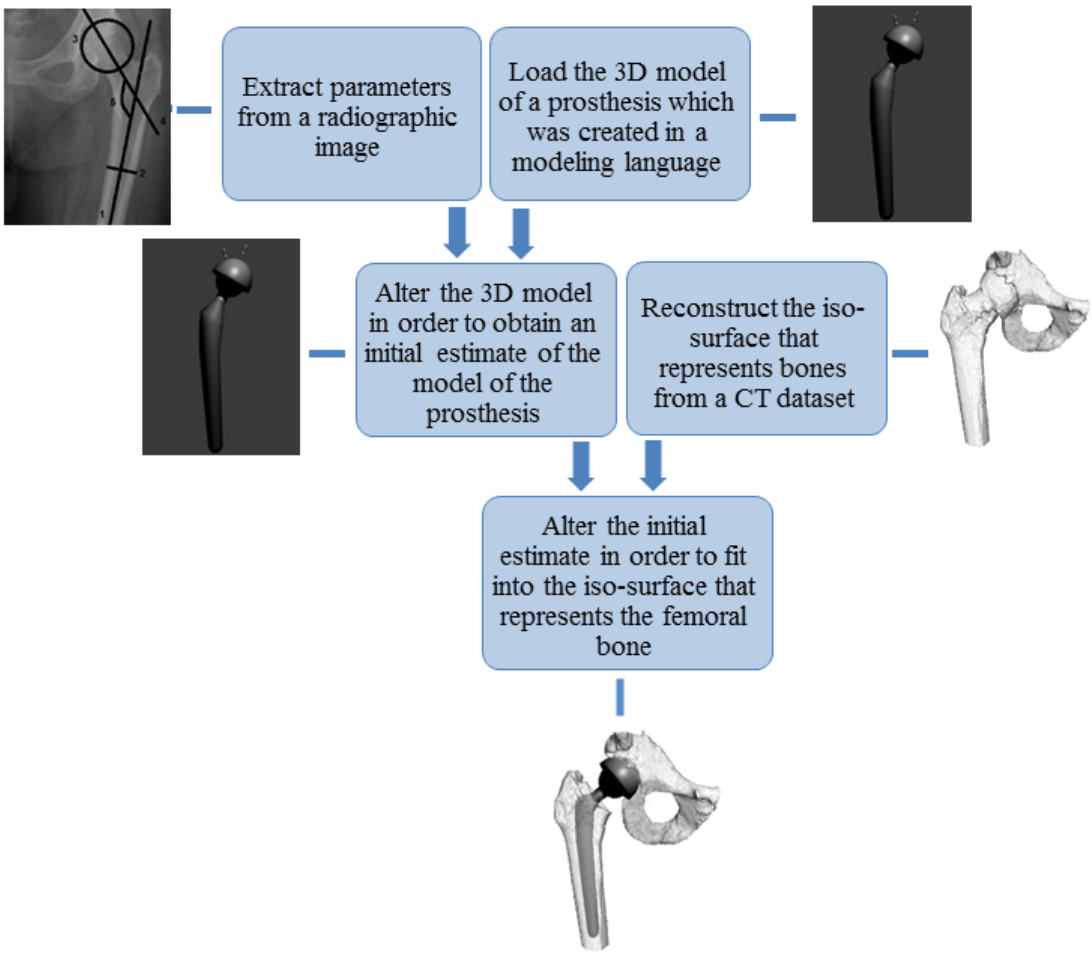

Fig. 4.1. Flow of the process that fits a 3D model of the prosthesis into an iso-surface that represents the femoral bone

Chapter 2 covers the first stage of the process. The parameters important for obtaining an initial 3D model of the prosthesis are briefly described. These parameters are highlighted in Fig. 4.2 which shows a part of an anterior-posterior orthopedic X-ray.

The outlined parameters are the following:

1. The diaphyseal axis
2. The bone width
3. The circle that approximates the femoral head
4. The femoral neck axis
5. The cervico-diaphyseal angle, which is the angle defined by the diaphyseal axis and the femoral neck axis

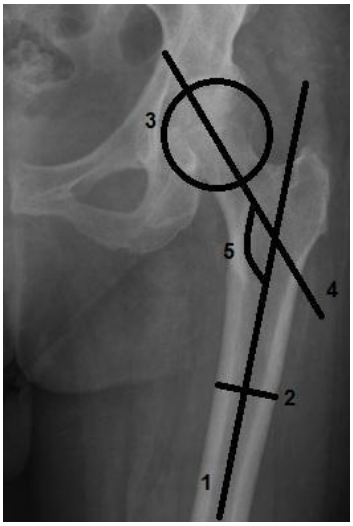

Fig. 4.2. Parameters important for obtaining an initial estimate of the 3D model of a prosthesis

If the value of the cervico-diaphyseal axis does not range between 125 and 135 degrees, part of the femoral bone needs to be replaced by a prosthesis. The extraction of the parameters important for this method is described in chapter 2.

Starting from a 3D model created in a modeling language like Blender or 3D Max, the dimensions of the prosthesis are altered to fit the parameters previously extracted. In our attempts, the 3D model of the prosthesis was altered by simple geometric transformations like translation, scaling or rotation. Also, in order to refine the transformations, we divided the prosthesis into three separate components, such that the transformations can be applied separately to each part. The prosthesis can be represented through a hierarchical structure composed of rigid links and joints. The rigid links are the head, the neck and the body of the prosthesis. The joints are the points that connect the rigid objects. The neck can be rotated relative to the head, and the body can be rotated relative to the neck of the prosthesis.

Fig. 4.3 depicts a model of a prosthesis generated with Blender.

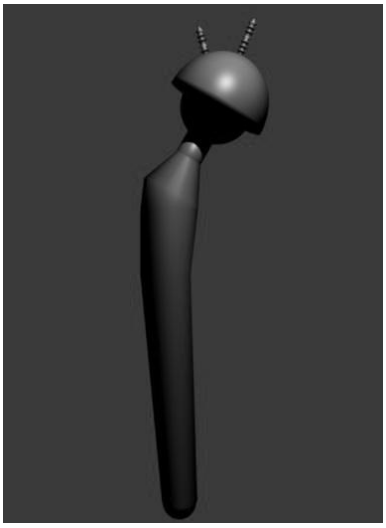

Fig. 4.3. The 3D model of a prosthesis in hip arthroplasty

The two generated triangular surfaces, the 3D model of the prosthesis and the 3D model of the bones, are combined for the goal of obtaining the final shape of the prosthesis. This phase requires human interaction to generate the most appropriate customized prosthesis. As previously mentioned, the prosthesis can be scaled, rotated or translated. Also, its separate components can be rotated within the articulated structure.

The steps of the method that were implemented are the following:

- Extraction of parameters of interest from a radiographic image
- Loading of a standard 3D model representing the prosthesis
- Iso-surface extraction with marching cubes algorithm
- User interaction tools for fitting the prosthesis in the iso-surface that represents the femoral and pelvic bones: only the transformations (rotation/scaling/translation) that are applied to the whole 3D model representing the prosthesis.

Even if the reconstruction of the femoral and pelvic bone can be parallelized with CUDA, as discussed in section 3.1, the process of fitting the prosthesis into the femoral bone is time consuming. The user interaction is intuitive, requiring only simple operations like translating/scaling/rotating the prosthesis or different components of the prosthesis. However, there is no computerized method that tests if the final model of the prosthesis fits the interior of the femoral bone. In consequence it can be concluded that the semi-automatic 3D fitting of the prosthesis requires a lot of user interaction, it is prone to errors and does not represent a considerable improvement to the hip arthroplasty field in the current form. This is the reason we propose in the following sections other methods, with very little human interaction, which contribute to the final goal of obtaining the 3D model of a customized prosthesis based on medical images.

4.2. 2D BONE SEGMENTATION BASED ON ACTIVE CONTOURS WITHOUT EDGES

The method proposed in the previous section is time consuming and does not guarantee, in its current stage, the correct fitting of the prosthesis inside the femoral bone.

In the previously proposed method, the 3D model of the prosthesis is generated based on the 3D model of the femoral bone, which is extracted as an iso-surface from a CT dataset. However, the low contrast between bones and other tissues makes it difficult to correctly extract the bones based on a single iso-value. Also, there is no possibility to distinguish between the femoral bone and the other bones (like the pelvic bones) extracted from the CT dataset.

In this section we propose a new algorithm, which segments bones from other tissues in CT slices and discriminates between different bones located within close proximity. This algorithm is used further for the 3D segmentation of the femoral bone that is described in section 4.3.

There are a lot of image segmentation techniques that try to differentiate between background (bone) and object (other tissues) pixels, but many of them fail to discriminate between different objects that are close to each other. Some characteristics of CT images like low contrast between background and foreground or inhomogeneity within the objects increase the difficulty of correctly segmenting images. We designed a new segmentation algorithm based on active contours without edges, a technique detailed in section 1.3.2. We also used other image processing techniques such as nonlinear anisotropic diffusion and adaptive thresholding in order to overcome the segmentation problems stated above. The new technique led to very good results, but the time complexity was a drawback. However, this drawback was significantly reduced with the use of GPU programming.

Section 1.3 discusses the state of the art in image segmentation and describes the advantages and disadvantages of each method relative to CT image processing. Each existing method meets some difficulties caused by the poor quality of the CT images. The new segmentation algorithm takes into account the imperfections of CT images or other images with low contrast, not only differentiating between objects and background, but between different objects.

The segmentation algorithms described in detail in section 1.3 were tested on ten noisy CT datasets. Fig. 4.4 presents an image before (a) and after being processed with 2D graph cuts (c), 3D graph cuts (d) and active contours without edges (e).

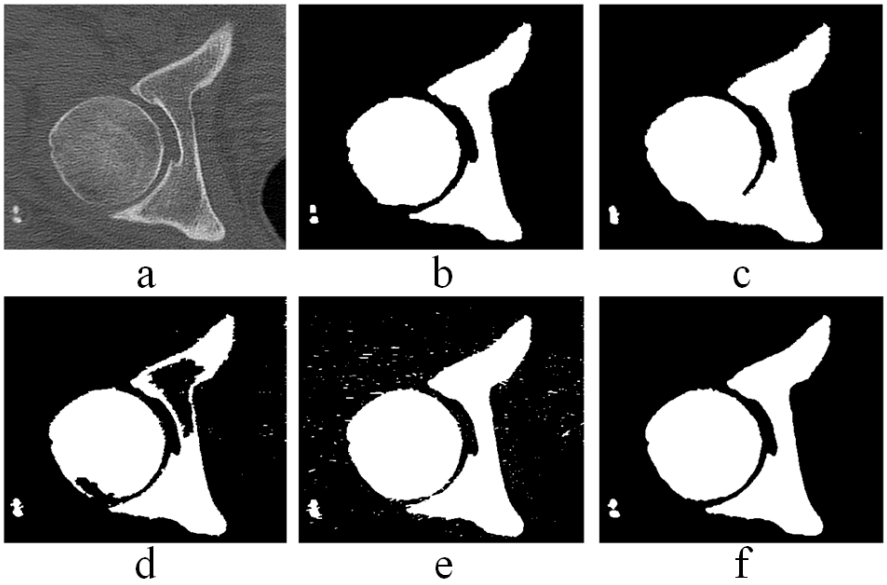

Fig. 4.4. Comparison of different CT image segmentation methods: (a) original image, (b) image segmented by a human specialist, (c) image segmented with 2D graph cuts, (d) image segmented with 3D graph cuts, (e) image segmented with active contours without edges (f) image segmented by the new proposed algorithm

Compared to the manual segmentation (b), the results show that, even if most pixels are correctly labeled as “object” or “background”, there are some problems with discriminating between different bones. Our algorithm segments the objects from the background quite well,

without connecting different objects. In the example provided in Fig. 4.4 it can be observed that the new proposed segmentation (f) was the closest to the output of the manual segmentation. Section 4.2.3 presents the results of a comparison with the other segmentation algorithms on ten datasets in order to demonstrate the quality of our algorithm in more detail.

The algorithm was tested and exemplified on CT images, but it can be applied on all kinds of grayscale images that have the following characteristics:

- the foreground has a higher intensity than the background;
- the images contain multiple objects that are positioned close to each other and should not be connected;
- there are big inhomogeneities within the foreground.

4.2.1. Image segmentation algorithm

This section presents the new segmentation technique. The steps of the algorithm are illustrated in Fig. 4.5 and are further described in detail.

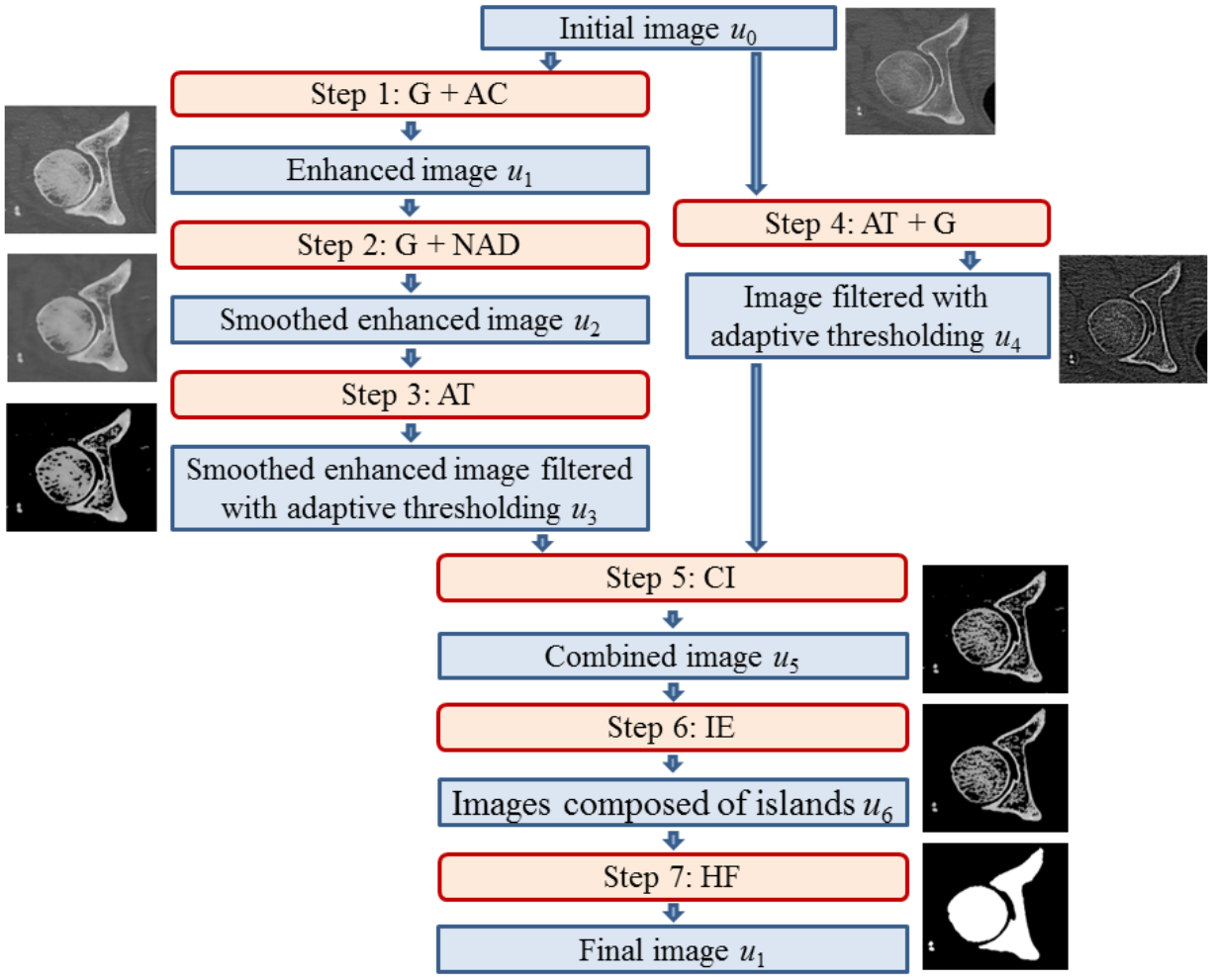

Fig. 4.5. Flow of the new image segmentation algorithm

Step 1: Gaussian filtering and active contours without edges (G+AC)

As previously mentioned, the new segmentation technique is based on the active contours model by Chan and Vese [11]. Here we state our reasons for making some changes to the original algorithm. The first change is the omission of the length term in equation (1.29). The original algorithm was tested on images where two different objects were positioned very close to each other. The output contained a single object, composed of the two initial objects. An explanation could be the one given by Chan and Vese regarding the use of the length term as a scale parameter. If the constant μ from (1.29) is small, then also smaller objects will be detected. If it is larger, then only larger objects are detected, or objects that are grouped together. We do not want different objects close to each other to be interpreted as a single object. Therefore, equation (1.29) is changed into the following one:

$$F(c_1,c_2,C)=\int_{\Omega}|u_0(x,y)-c_1|^2 H(\phi(x,y))dxdy+ \int_{\Omega}|u_0(x,y)-c_2|^2(1-H(\phi(x,y)))dxdy \quad . \tag{4.1}$$

As mentioned in section 1.3.2, the active contours algorithm without edges follows an iterative method. Knowing $\phi(x,y,s)$ at time $t_s=s\cdot\Delta t$, $c_1(s)$ and $c_2(s)$ can be determined by using (1.32) and (1.33). Then, $\phi(x,y,s+1)$ can be computed based on the following discretization and linearization of (1.34) in ϕ:

$$\phi(x,y,s+1)=\phi(x,y,s)+\Delta t\cdot\delta_{\varepsilon}(\phi(x,y,s))\cdot [-(u_0(x,y)-c_1(s))^2+(u_0(x,y)-c_2(s))^2], \tag{4.2}$$

where Δt denotes the time step size and $u_0(x,y)$ represents the initial image.

Most of the tested images had a lot of noise. In order to remove this noise the initial image u_0 is smoothed with a Gaussian filter. The image segmentation that divides the pixels based on the value of ϕ relative to 0 is too rough in the sense that it leads to a binary image, where white pixels belong to the foreground and black pixels to the background. The original active contours approach without edges also requires a large number of iterations to get to a stable configuration, i.e., the final segmentation. An alternative to converting the image into a binary segmented one could be a segmentation where the active contours model represents only an enhancement step.

After computing the values of ϕ based on (4.2), in the last iteration the values of ϕ are normalized to [-1,1]. In the original active contours model without edges, if $\phi<0$, then the segmented image $u_1(x,y)=0$ and if $\phi\geq 0$, $u_1(x,y)=I_{\max}$. The biggest change to the original

algorithm is to compute the output image by normalizing ϕ to $[0, I_{\max}]$, where $I_{\max}$ is the maximum level of intensity:

$$u_1(x, y) = \frac{(1+\phi(x, y)) \cdot I_{\max}}{2}. \tag{4.3}$$

With the new approach, the output image u_1 has shades of gray, with an enhanced contrast between foreground and background. Fig. 4.6(b) presents a CT image after applying the first step from the proposed method.

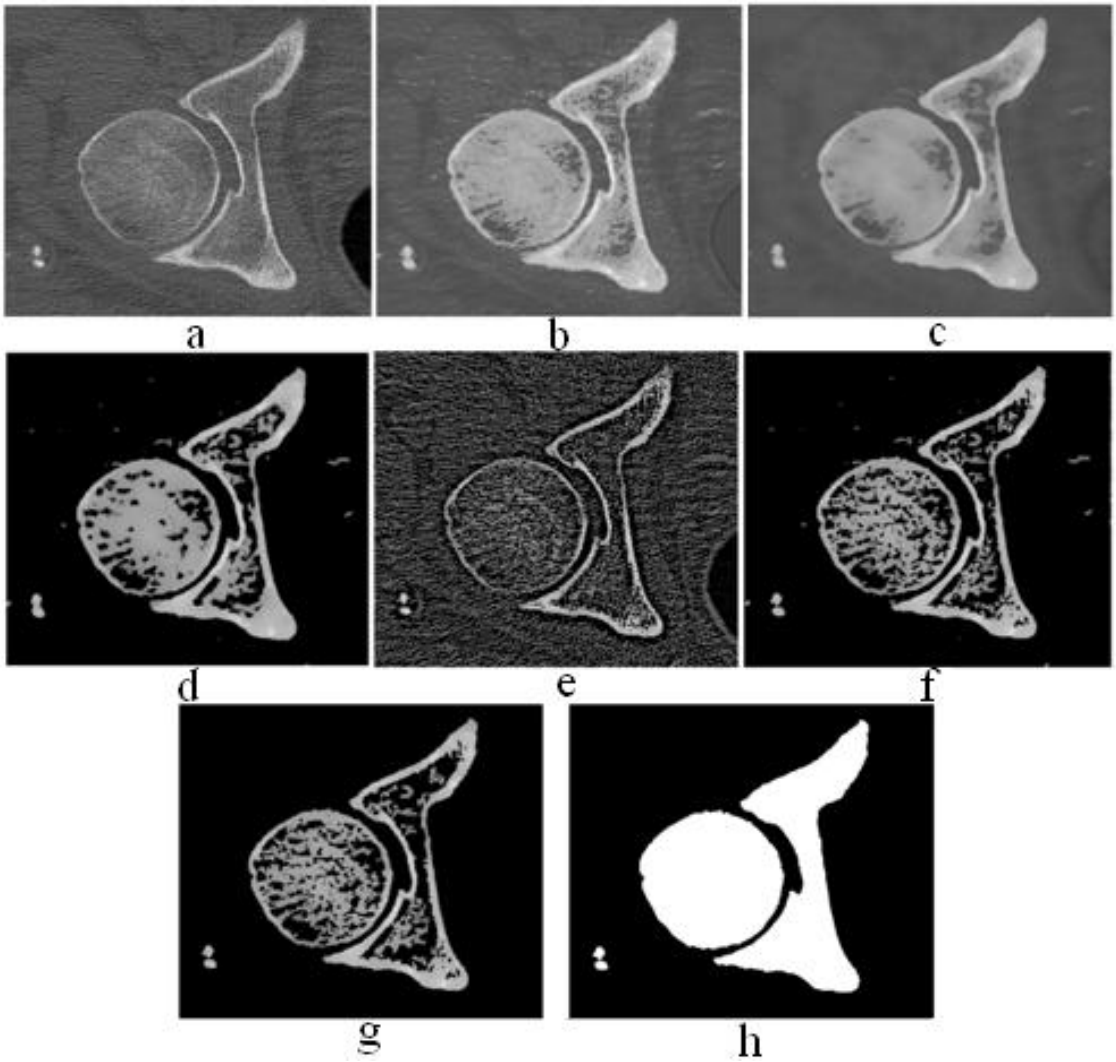

Fig. 4.6. Output images in the proposed image segmentation steps: (a) original image, (b) image enhanced with active contours without edges (Step 1), (c) image smoothed with nonlinear anisotropic diffusion (Step 2), (d) result of adaptive thresholding (Step 3), (e) output after applying an adaptive threshold on the original image (Step 4), (f) image obtained by combining images u_3 and u_4 (Step 5), (g) image resulting from connecting the foreground pixels into slice islands and removing those islands with low intensity (Step 6), (h) final image after applying the hole filling step (Step 7)

Step 2. Gaussian filtering and nonlinear anisotropic diffusion (G+NAD)

The Gaussian filter is applied on image u_1 to remove small discontinuities within the object, especially near the borders. The image has to be further smoothed, but without connecting different objects. A nonlinear anisotropic diffusion filter, which is detailed in section 1.1.2, is applied to the image, in order to reduce noise while preserving edges. Fig. 4.6(c) presents the output image u_2 after applying these two filters.

Step 3. Adaptive thresholding (AT)

Here we state the reasons for introducing another processing step, an adaptive thresholding. The intensities of the image u_2 are between 0 and $I_{\max}$. A simple segmentation would separate the pixels based on their value relative to $I_{\max}/2$. But there are two problems with this segmentation:

- Some pixels with intensity close to the threshold $I_{\max}/2$ but lower than $I_{\max}/2$ should belong to the foreground. The interval to search for these pixels is set to $[I_{\max}/2-\Delta I, I_{\max}/2]$, where ΔI is a fixed parameter

- There are some pixels whose intensities are close to the threshold $I_{\max}/2$ but greater than $I_{\max}/2$ which should belong to the background. The interval to search for these pixels is set to $[I_{\max}/2, I_{\max}/2-\Delta I]$.

The solution to these problems is the use of two thresholds $T_1 = I_{\max}/2-\Delta I$ (low) and $T_2 = I_{\max}/2+\Delta I$ (high). All the pixels with intensity value lower than T_1 are labeled as background pixels (their intensity is set to 0). All the pixels with intensity value greater than T_2 are considered foreground pixels but their intensity remains unchanged. An adaptive threshold is applied to all the pixels that belong to the interval $[T_1, T_2]$. The filter removes pixels that could wrongly connect different objects. Choosing a window of neighbors W_p of size $n \times n$ for the current pixel p we compute the average intensity, multiplied by a fixed parameter α:

$$M = \frac{\sum_{q \in W_p} u_2(q)}{n^2} \cdot \alpha . \tag{4.4}$$

M is a threshold that divides the pixels into foreground and background. If $u_2(p) < M$, then p is labeled as background pixel. If $\alpha = 1$, there are still some pixels wrongly considered as part of the foreground. This is why we set a slightly higher threshold with $\alpha = 1 + \Delta\alpha$, where $\Delta\alpha > 0$ is very close to 0. Fig. 4.6(d) illustrates the result u_3 of applying the adaptive thresholding to image u_2.

Step 4. Adaptive thresholding and Gaussian filtering on initial image (AT+G)

Due to the third step, there are fewer wrongly labeled pixels. However, in case of objects that are very close to each other, or small contrast between foreground and background, there could still be pixels that connect different objects. The solution to this problem is to apply another adaptive threshold, but this time, on the initial image u_0. The output image is also smoothed with a Gaussian filter. This adaptive thresholding is applied to all the image pixels, not only to those within the interval $[T_1,T_2]$. Applying the second adaptive thresholding assures the removal of the pixels that could have been wrongly labeled as "object" because of the smoothing steps – the two Gaussian filters and the nonlinear anisotropic diffusion. Fig. 4.6(e) presents the output image u_4 after applying the adaptive threshold on the initial image u_0.

Step 5. Combining images (CI)

The next step combines the images u_3 and u_4 to obtain an image with very few or no pixels that connect different objects. For the current pixel p, if $u_3(p)>0$ and $u_4(p)<T_3$, where T_3 is an experimentally determined threshold, then the combined image $u_5(p)=0$, otherwise $u_5(p)=u_3(p)$. Fig. 4.6(f) shows the output image u_5 after combining images u_3 and u_4.

Step 6. Island extraction (IE)

The low value of the threshold T_2 from Step 3 removes small discontinuities within the objects. However, this also adds wrongly labeled foreground pixels. This issue is handled in the following manner: the foreground pixels that are connected are grouped into islands. For every island, the average intensity I_{avg} is computed. If $I_{avg}<T_4$, where T_4 is a fixed parameter, then the pixels of the current island are labeled as belonging to the background . This step assures that only those low intensity pixels that are connected to a large number of high intensity pixels are considered to belong to the foreground. Fig. 4.6(g) presents the output image u_6 after removing the low intensity islands from image u_5.

Step 7: Hole filling (HF)

The last step in the segmentation process solves the problem of inhomogeneities within the object. Starting from the first encountered pixel in the image, in a left-write top-bottom traversal, we do a breadth first search (BFS) in order to visit all the background pixels connected to the first one. All the other pixels that have not been visited in the BFS are considered to be part of the foreground, as depicted in Fig. 4.6(h). This step is called "hole filling" because it re-labels all the background pixels that are located inside a closed foreground island. The problem of this step is that it does not differentiate between inhomogeneities within an object and real cavities. This drawback can be overcome in the following manner: if there is the certainty that a pixel belongs to a real cavity, and not an inhomogeneity within the object, this pixel can be set as another seed point for the BFS that searches for all the connected background pixels.

The pseudo code of the proposed segmentation algorithm is given below.

Pseudo code 4.1. 2D Image segmentation algorithm

```
Step 1.Gaussian filtering + modified active contours without edges
for every slice s in volume v0, do (the volume is obtained from CT dataset)
        apply Gaussian filter on slice s
        initialize ϕ (ϕ is the Lipschitz function that defines the curve C)
        for every iteration it, do
                compute average values c1 and c2 based on (1.32) and (1.33)
                for every pixel p in slice s, do
                        compute image force F based on (4.1)
                        compute ϕ based on normalized image force with (4.2)
                end for
                if it is the last iteration, then
                        for every pixel p in slice s, do
                                normalize ϕ to [-1,1]
                        end for
                end if
        end for
end for (the result is volume v1)
Step 2. Gaussian filtering + nonlinear anisotropic diffusion
for every slice s in volume v1, do
        apply Gaussian filter on slice s
        for each pixel p in slice s, do
                compute horizontal and vertical gradient for pixel p
                compute eigenvectors of p, based on (1.10)
                compute eigenvalues of p, based on (1.11)
                compute diffusion tensor D
        end for
        for every pixel p, do
                compute new value of pixel p based on (1.13)
        end for
end for (the result is volume v2)
Step 3. Adaptive thresholding
for every slice s in volume v2, do
        for every pixel p in slice s do
                if value(p)<T1 then
                        p is background pixel (value(p)=0)
                else
                        if value(p)<T2, then (apply adaptive thresholding)
                                compute M based on (4.4)
                                if value(p)<M then
```

```
                                        p is background (value(p)=0)
                                    end if
                                end if
                            end if
                        end for
end for (the result is volume v3)
Step 4. Adaptive thresholding and Gaussian on initial image
for every slice s in volume v0, do
        for every pixel p do
                compute M based on (4.4)
                if value(p)<M then
                        p is background (value(p)=0)
                end if
        end for
        apply Gaussian filter on slice s
end for (the result is volume v4)
Step 5. Combine images from volumes v3 and v4
for z=0 to maxZ, do (maxZ is the number of slices in the volume)
        for every pair of coordinates (x,y), do
                if v3(x,y,z)>0 and v4(x,y,z)<T3 then
                        v5(x,y,z)=0  (v5 is the output volume)
                else
                        v5(x,y,z)=v3(x,y,z)
                end if
        end for
end for
Step 6. Island extraction
for every slice s in volume v5 do
        for every foreground pixel p that has not been visited do
                BFS(p) (visit all the object pixels that are connected with p)
                compute average intensity Iavg of current island Isl
                if Iavg<T4 then
                        remove island Isl (all its pixels are labeled as background pixels)
                end if
        end for
end for (the result is volume v6)
Step 7.Hole filling
for every slice s in volume v6 do
        determine first background pixel p
        BFS(p) (visit all the background pixels that are connected with p)
        for all background pixels p that have not been visited in the BFS, do
                label p as foreground pixel
        end for
```

```
end for
End of pseudo code
```

4.2.2. Parallel implementation using CUDA

The segmentation was implemented both on the CPU and on the GPU. Even for relatively small datasets, the CPU implementation takes a lot of time. On the other hand, the CUDA architecture [28] provides the possibility of running the same instructions for each pixel in a parallel manner, considerably improving the computing times.

The parallel implementation of the Gaussian filtering with CUDA is based on the Sobel filter from the CUDA SDK example [29] and is described in section 2.3.1. The method is straightforward, because every pixel is computed by a CUDA thread based on the pixels in a 3×3 neighborhood.

The modified active contours algorithm without edges is also implemented using the GPGPU technology in a similar manner as the one explained by Bojsen-Hansen [33]. Here there are some operations that are not intuitively parallel. The computation of the average background and foreground values c_1 and c_2, the normalization of the image force F and the Lipschitz function ϕ are detailed below because they are slightly modified from the implementation by Bojsen-Hansen. [33] describes the parallel implementation of these operations for a single image, while our proposed implementation handles volumes or stacks of images.

Let S be the total number of slices in the volume, and w and h be the width, respectively the height of a slice. The implementation of the steps that compute the variables c_1, c_2, F and ϕ is based on the "reduction" application from the CUDA SDK example [29]. The reduction method is exemplified on the normalization of ϕ, which requires the maximum value of ϕ for each image.

The computation of the maximum value of parameter ϕ is not intuitively parallel, because it depends on the values of ϕ for all the pixels in an image. We divide the method into two stages. The first stage computes the maximum value of ϕ for the pixels within each row in an image. The second one computes the maximum value of ϕ for all the rows in the image, i.e., the final maximum value of ϕ for an image. The algorithm computes a maximum value of ϕ for each image in the volume. The flow of the operations and the structuring of the data are depicted in Fig. 4.7.

As discussed in section 2.3.2.2, the CUDA threads are organized into blocks and grids. In the first stage the threads are structured into one-dimensional blocks of size w and the blocks are structured into a two-dimensional grid of size (h, S). Each block handles a row in an image. The threads in a block cooperate in order to determine the maximum value of ϕ for the current row. Each block loads the data, namely the values of ϕ, into shared memory, into an array of size w. The computation of the maximum value for the current block is done in $\log_2(w)$ iterations. In each iteration, the number of active threads is divided by 2, as illustrated in Fig. 4.8.

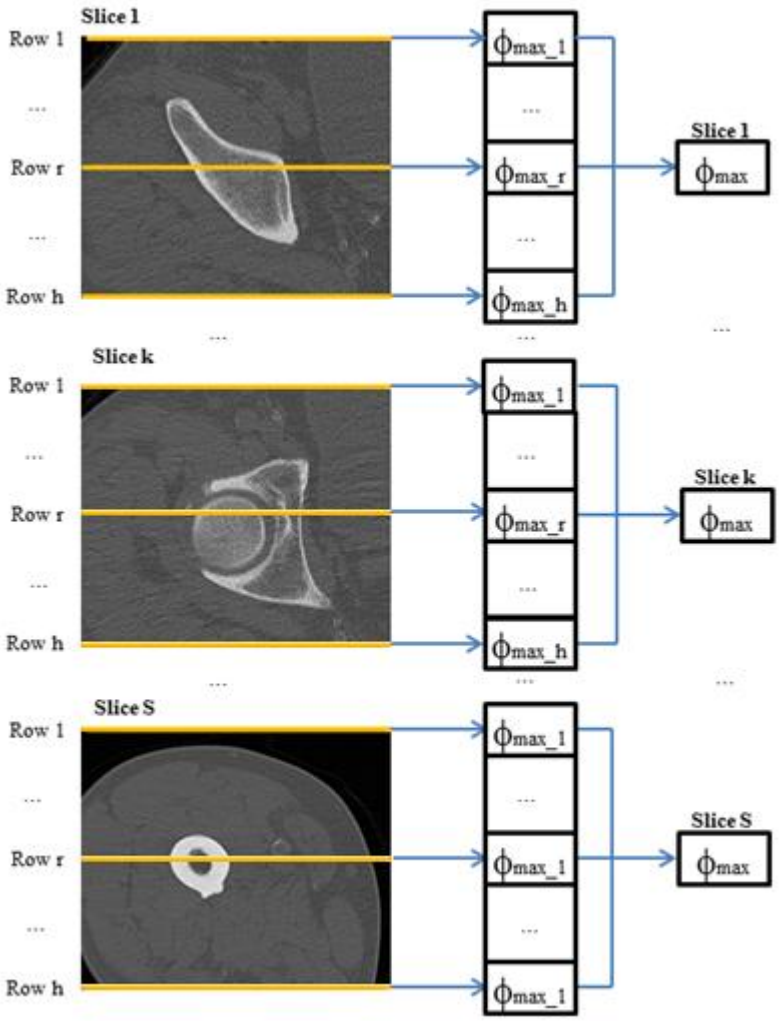

Fig. 4.7. Computation of maximum value of ϕ for each image: in the first stage the algorithm determines the maximum for each row in an image and in the next stage computes the maximum for the whole image

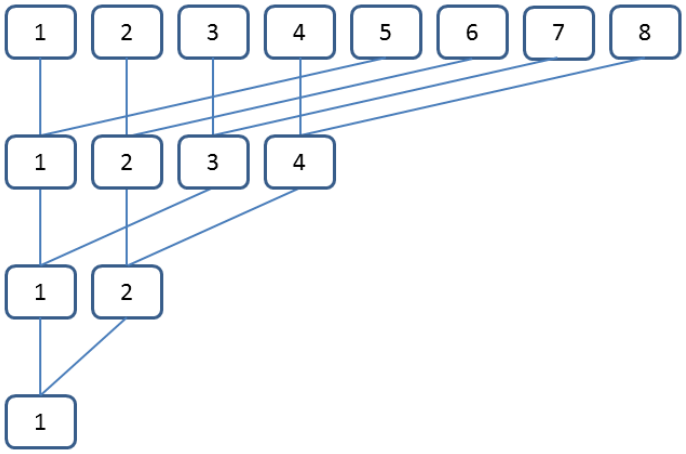

Fig. 4.8. Reduction method

The pseudo code of the CUDA kernel that computes the maximum value of ϕ for a row in an image is given pseudo code 4.2.

Pseudo code 4.2. Computation of the maximum value of ϕ for a row y in slice z

```
CUDA kernel that computes the maximum value of ϕ based on the reduction method
determine coordinates x, y and z of the current pixel (based on the thread ID and the block ID)
copy values of ϕ into shared memory in vector ϕ_max
synchronize threads
for s=w/2 to 0; s=s/2, do (s is the stride)
        In each iteration, the number of active threads is divided by 2)
        if x<s then  (only half of the threads run the comparison)
                if ϕ_max[x]<ϕ_max[x+s] then
                        ϕ_max[x]=ϕ_max[x+s]
        end if
        synchronize threads
end for
Only the first thread writes the result to global memory
If x=0 then (if the current thread is the first one)
        ϕ[y][z]=ϕ_max[0] (the maximum value in shared memory is located in the first field of ϕ_max)
End of pseudo code
```

The second stage is similar to the first one, but with a different structuring of the CUDA threads. The algorithm computes the maximum of each image in the volume based on the maximum of each row in that image. The threads are structured into one-dimensional blocks of size h while the blocks are organized into a one-dimensional grid of size S .The threads in a block cooperate in determining the maximum value of ϕ for a slice.

The parallel implementation of the nonlinear anisotropic diffusion filter is based on the Sobel filter from the CUDA SDK Example [29].

Another algorithm that is not intuitively parallel is the hole filling step, i.e., the breadth first search that extracts all the background pixels that are connected to the first encountered background pixel. The CUDA implementation is based on the method proposed by Harish and Narayanan [34]. They describe techniques to accelerate large graph algorithms on the GPU using CUDA. We adapt their implementation of the BFS operation on graphs to images. The algorithm keeps for each image two Boolean arrays, visited and frontier, of the same size as the image size. The first array stores information about the pixels that have been visited in the breadth first search. The second array stores information about the active pixels, the pixels that have been visited, but with neighboring pixels that have not been processed yet. Another Boolean variable, finish, is set to false while there are still pixels that have not been visited.

The pseudo code of the algorithm gives more details about the initialization and update of the arrays.

Pseudo code 4.3. Hole filling step

```
Initialize visited and frontier arrays
for all pixels p in the set of images, do (in parallel with a CUDA kernel)
        determine coordinates (x,y) based on the thread and block ID
        if x=0 or y=0 or x=w or y=h, then (all pixels from the image border are part of the frontier; this
way, the algorithm converges faster than if we initialize only the first encountered background pixel)
                frontier[x][y]=true
        else
                frontier[x][y]=false
        end if
        visited[x][y]=false
end for
Do BFS - iterative method
while finish=false, do (there are still pixels that have to be visited)
        finish=true
        for every pixel p in the dataset do (in parallel with a CUDA kernel)
                determine x and y coordinates
                if frontier[x][y]=true (p is one of the currently processed pixels)
                        frontier[x][y]=false
                        visited[x][y]=true
                        check neighbors in a 4-neighborhood
                        if there is at least one neighbor of p that is background pixel, do
                                finish=false; (there is no problem with memory read/write hazards; it is
enough for one thread to change it to false. It means that there are still pixels to be visited)
                        end if
                end if
        end for
end while
All the background pixels that have not been visited in the BFS are considered to be foreground pixels
…
End of pseudo code
```

Table 4.1 shows all the processing steps that were implemented in CUDA. It also provides a comparison between the running times on a dataset of 256 images (each of size 512^2) for the implementation on the CPU and on the GPU. We set the size of the Gaussian filters to 9x9 and the size of the neighborhood window for the adaptive thresholds to 21x21. The maximum number of iterations for active contours without edges is 50 and the maximum number of iterations for nonlinear anisotropic diffusion is 20. The tests were made on an i7-2600K 3.40 GHz processor with 8GB RAM and an Nvidia GeForce GTX 590 GPU card with 1.5 GB RAM. The island extraction and the hole filling step are both variations to the BFS algorithm, and are based on recursion. Even if the hole filling step was implemented with CUDA, the island extraction was not approached in a parallel manner. The reason is the small size of an extracted foreground island, as compared to the number of background pixels in a slice. CUDA

implementations are practical if we need to process large data or do the same massive computation for each element in the data. Thus, the island extraction step, which processes a small number of pixels for each island and is not compute intensive, does not represent an algorithm suitable for parallelization.

Table 4.1. Computing times for the new bone segmentation algorithm on the CPU and the GPU

Step in the algorithm	Time on the CPU (sec)	Time on the GPU (sec)	Performance gain
Step 1: G + AC	2245.71	4.02	558x
Step 2: G + NAD	340.19	2.85	119x
Step 3: AT	11.78	0.39	30x
Step 4: AT + G	205.98	1.21	170x
Step 5: CI	0.14	0.1	1.4x
Step 7: HF	59.02	2.7	22x
All steps	2862.82	11.49	249x

The CUDA implementation has a great impact on the speed of the segmentation process. Thus, with the help of the GPGPU paradigm we now have a quite fast algorithm. The next section shows how accurate this algorithm is.

4.2.3. Results

The new segmentation algorithm was tested on ten noisy CT datasets with different number of slices, each of size 512x512. The parameters from section 4.2.1 were the same for all the images: $\Delta I = 30 => T_1 = 97.5$ and $T_2 = 157.5$; $\Delta\alpha = 1/29 => \alpha \approx 1.0345$; $T_3 = 50$ and $T_4 = 110$. The initial curve C is a circle located in the center of the image with a radius of 100: $\phi = -\sqrt{(x-256)^2 + (y-256)^2} + 100$.

The proposed implementation was compared with three other segmentation methods:

- Active contours without edges, described in section 1.3.2, with the following parameters: the maximum number of iterations for computing ϕ is 100, and the parameters from (1.29) are $\eta_1 = 1$, $\eta_2 = 1$, $\mu = 0.2 \cdot 255^2$ and $\nu = 0.01 \cdot 255^2$.

- The segmentations using 2D and 3D graph cuts, described in section 1.3.1, with the following parameters: $k = 3$, $v = 5$ and $w = 3$.

Depending on the difficulty of correctly labeling the foreground (bones) and the background pixels (other tissues), the CT datasets were divided into two categories: low and high difficulty. Fig. 4.9 shows a slice from each dataset. The images segmented by the four algorithms were compared with the segmentation made by a human specialist. The tests consisted of counting the correctly labeled pixels, the foreground pixels that were wrongly labeled as background pixels (false negatives), the background pixels that were wrongly labeled as foreground pixels (false positives), and the number of images where two different objects were wrongly connected. If F_1 is the number of correctly labeled foreground pixels, and F_2 is the

number of false negatives, then the false negative percentage is $E_F = F_2/(F_1 + F_2)\cdot 100$. Similarly, the false positive percentage is $E_B = B_2/(B_1 + B_2)\cdot 100$, where B_1 is the number of correctly labeled background pixels and B_2 is the number of false positives. Table 4.2 presents the false positive and the false negative error for the low difficulty datasets.

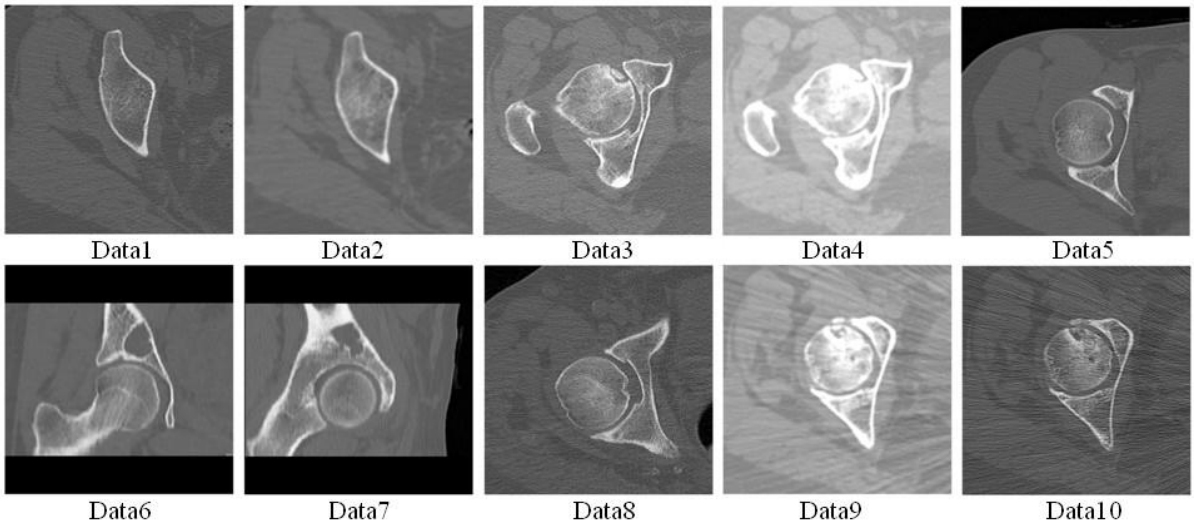

Fig. 4.9. Slices from the datasets that were tested: low difficulty datasets (first row) and high difficulty datasets (second row)

Table 4.2. Errors in labeling pixels for the low difficulty datasets

Algorithm	Data1		Data2		Data3		Data4		Data5	
	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)
Our alg.	0.028	0.691	0.068	0.651	0.56	0.897	0.247	1.214	0.922	1.342
2D GC	0.015	0.497	0.0337	0.676	13.724	0.677	23.905	0.663	0.258	1.584
3D GC	0.068	0.772	0.11	0.811	4.564	1.525	12.029	11.332	0.927	1.116
AC	0.015	0.585	0.027	0.814	0.861	0.814	0.128	0.434	0.859	0.818

Table 4.3 presents the percentage of images where two different objects were wrongly connected, for the low difficulty datasets.

Table 4.3. Percentage of images where different objects were wrongly connected for the low difficulty datasets

Algorithm	Percentage of images (%)				
	Data1	Data2	Data3	Data4	Data5
Our algorithm	0	0	0	10.42	0
2D graph cuts	0	0	4.16	14.58	34.09
3D graph cuts	0	0	8.33	29.16	13.63
Active contours	0	0	8.33	18.75	21.59

The high difficulty datasets have more noise, more inhomogeneities within the objects, and a bigger change in intensity between different images of the same dataset. Table 4.4 presents a comparison regarding the false positive and false negative errors between our algorithm and the other segmentation algorithms, for the high difficulty datasets.

Table 4.4. Errors in labeling pixels for the high difficulty datasets

Algorithm	Data1		Data2		Data3		Data4		Data5	
	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)	E_B (%)	E_F (%)
Our alg.	0.999	1.163	1.441	1.342	2.79	0.369	6.331	1.569	6.83	1.76
2D GC	4.59	1.219	0.361	2.629	2.924	0.847	4.573	21.75	8.276	6.653
3D GC	2.396	1.689	1.147	1.631	3.404	0.505	10.774	8.96	23.748	3.138
AC	0.752	2.037	0.45	1.725	3.533	0.405	2.977	1.937	6.2	4.275

Table 4.5 presents the comparison between the segmentation algorithms regarding the percentage of images were different objects are wrongly connected, for the high difficulty datasets.

Table 4.5. Percentage of images where different objects were wrongly connected for the high difficulty datasets

Algorithm	Percentage of images (%)				
	Data1	Data2	Data3	Data4	Data5
Our algorithm	0	0	0	21.73	8.69
2D graph cuts	10	100	10.46	69.56	86.95
3D graph cuts	30	30	6.58	73.91	30.43
Active contours	60	90	10.85	100	91.30

The results in computing the pixel labeling error for our algorithm were comparable to or even better than the other algorithms' results. The big difference can be observed in the percentage of images where different objects were wrongly connected. Our algorithm discriminated quite well between different objects, even on noisy images. This does not hold for the other algorithms. In order to obtain an overview of the differences between the four segmentation algorithms, we computed, for all the datasets, the average of the false negative and false positive error in labeling the pixels, and the average percentage of images where different objects were wrongly connected. These differences can be observed in Fig. 4.10.

Even if the average error in pixel labeling from the active contours without edges segmentation is comparable to the results obtained with our algorithm, the average percentage of images where different objects are wrongly connected shows that our implementation is superior. From the tests described in this section and from Fig. 4.10 we can draw the conclusion that the new segmentation algorithm is 98% and 99% accurate regarding the labeling of background and foreground pixels, respectively, and 96% accurate in discriminating between different objects in CT images. Also, it can be observed that in seven cases out of ten, the proposed algorithm differentiated perfectly between objects. The percentage of slices where different objects were wrongly connected is 0% in these cases.

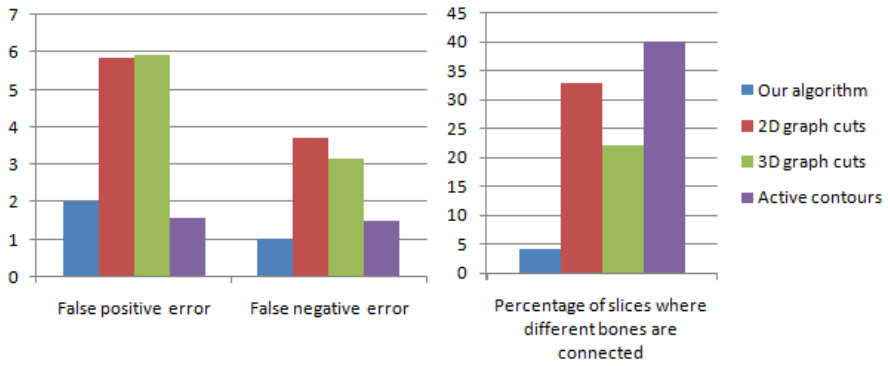

Fig. 4.10. Comparison between our algorithm and other segmentation algorithms regarding the average of the false positive and false negative errors in pixel labeling and the average percentage of images where different objects were wrongly connected

The next step in the 3D processing of CT images in hip replacement is to connect segmented CT images in order to obtain the whole volume occupied by bones.

4.3. 3D BONE SEGMENTATION

This section proposes a new method for the 3D segmentation of bones in CT images and the reconstruction of the femoral bone. The proposed method can highly reduce the time spent for the planning of the surgeries in hip arthroplasty.

The ultimate goal of our research is to develop a software system capable to automatically generate the 3D model of a personalized prosthesis at the level of the hip, starting from a stack of CT images. One of the steps is the extraction of the volume occupied by the femoral bone and the pelvis.

Section 4.1 proposes a semi-automatic method for the 3D simulation of fitting prostheses inside femoral bones. But the user interaction is time consuming. This is why we propose a fully automated pipeline for the final goal of obtaining personalized implants.

The first problem is the segmentation of bones from other tissues. The poor quality of the CT images poses some difficulties to the existing segmentation algorithms. Section 4.2 describes a new segmentation algorithm based on active contours without edges. This algorithm takes into account all of the characteristics of CT images in order to differentiate between bones and other tissues, but also between different bones. However, the 2D segmentation algorithm can be further improved based on the relation between adjacent slices in the CT dataset.

4.3.1. Bone Reconstruction Algorithm

The proposed method of bone reconstruction from CT images is composed of two stages: the segmentation of the bones in CT images and the 3D reconstruction from the segmented

images. The input of the bone reconstruction stage is composed of the binary segmented CT images into foreground (bone) and background (other tissues) pixels. Even if any bone segmentation algorithm can be incorporated in this method, we use the technique presented in the previous section, slightly altered. Fig. 4.11 depicts the steps of the bone reconstruction algorithm.

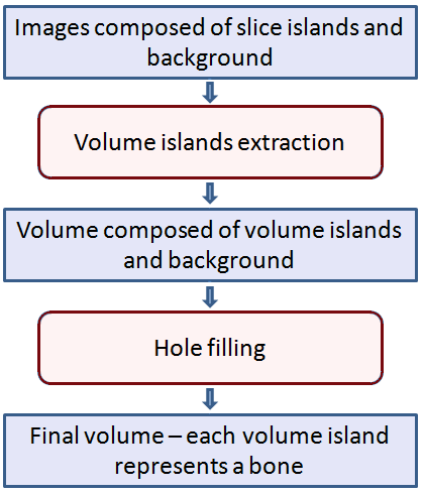

Fig. 4.11. Flow of the bone reconstruction stage

The last step in the segmentation phase, the hole filling, has a disadvantage that was discussed in section 4.2.1. The method starts a BFS search from the first encountered background pixel and visits all the background pixels that are connected to the first one. All the other background pixels that are not visited in the BFS are considered as being part of the object. This step re-labels all the background pixels that are surrounded by slice islands, either representing inhomogeneities within the foreground or real cavities inside the object. A possible solution for differentiating between inhomogeneities and cavities is given in 4.2.1. If there is the certainty that some pixels belong to real cavities, and not to inhomogeneities within the object, these pixels can be set as other seed points for the BFS that searches for all the connected background pixels. These seed points could be set based on the intensity. However, in CT images, the difference in intensity between inhomogeneities and cavities is sometimes insignificant. Fig. 4.12(a) provides such an example. As can be observed, the bone marrow has a similar intensity with the tissue that is located between the femoral bone and the pelvis.

Another solution would be a user interaction scenario where the doctors would choose the seed points belonging to the cavities. But the user interaction is time consuming. We propose an automatic solution to this problem that moves from the processing of each slice individually to a technique that takes into account the relation between slices.

The algorithm proposed in section 4.2 is altered by the removal of the hole filling step. Thus, the input for the bone reconstruction phase is the segmented dataset, composed of the slice islands and the background. An example of a slice from an input dataset is given in Fig. 4.6(g).

Fig. 4.12. The problem of the hole filling step: (a) initial image, (b) output image after applying the hole filling step from the segmentation proposed in [5] (c) correct segmentation obtained by applying the hole filling step in the bone reconstruction phase

Step 1.Volume islands extraction

Slice islands from adjacent slices, belonging to the same objects, are combined into volume islands. A volume island consists of all the slice islands from the dataset that are connected through neighboring slice islands. We further describe the concept of neighboring islands.

Let SI_1 and SI_2 be two slice islands from adjacent images in the CT dataset and let $size(SI)$ be the number of pixels belonging to a slice island SI. Two foreground pixels from adjacent slices $m_1(x_1, y_1, z_1)$ and $m_2(x_2, y_2, z_2)$ are neighbors if and only if:

$$\begin{cases} x_1 = x_2 \\ y_1 = y_2 \\ |z_1 - z_2| = 1 \end{cases}. \tag{4.5}$$

Let $connection(SI_1, SI_2)$ be the number of neighboring pixels pairs (m_1, m_2) where $m_1 \in SI_1$ and $m_2 \in SI_2$. We have experimentally determined a rule for deciding whether two slice islands are neighbors. If

$$connection(SI_1, SI_2) > \min(size(SI_1), size(SI_2)) \cdot \beta \ , \tag{4.6}$$

Where $0 < \beta < 1$ is a fixed parameter, then SI_1 and SI_2 are considered neighboring slice islands. In our tests we have chosen the value of $\beta = 0.25$. This value is big enough to ensure that islands from different objects are not wrongly connected, and small enough to ensure that islands belonging to the same object are considered neighbors.

The next step in the workflow is to delete the volume islands with the size (number of pixels) less than a given threshold T. T is chosen so that only the volume islands composed of

noise pixels are deleted. In our tests the variable T was set to 10000. This does not erase bone tissue, since the bone volume islands are composed of tens of thousands of pixels.

Step 2.Hole filling

After identifying all the volume islands, the hole filling step can be applied on each slice individually. First we do a breadth first search starting from a background pixel. We assume, without loss of generality, that the seed point of the BFS is a background pixel that does not belong to the interior of a slice island. All the background pixels that were not visited in the BFS are further inspected. The connected background pixels that were not visited form background islands. A background island is considered to be connected with a slice island if it has at least one pixel that is a direct neighbor of a pixel from that slice island. If a background island is connected with only one slice island, then its pixels are re-labeled as foreground pixels. This condition ensures that pixels belonging to other tissues besides bone do not connect different volume islands representing bones. The extraction of a background island and its inspection requires another BFS, starting from the first encountered pixel that belongs to the island.

The pseudo code of the proposed algorithm is given below.

Pseudo code 4.4. 3D segmentation algorithm

```
Stage 1. Volume island extraction
Each island slice on the first slice represents an island volume in the final volume
for each slice island si in the first slice (z=0) do
        volume[si]=current_volume_id (current_volume_id is the number of used id-s for the volume
                                      islands)
        volume_size[current_volume_id]=island_size[si]
        (volume_size stores the size of each volume island)
        (island_size stores the size of each slice island)
end for
for each slice z in the volume do (minus the first one)
        initialize neighbors vector (stores the number of neighbors for each pair of islands from
                                     adjacent slices)
        for each pair (x,y) of coordinates do
                if slice_island_id[x,y,z]>0 and slice_island_id[x,y,z-1]>0 then
                (slice_island_id stores for each pixel the id of the slice island it belongs to; if the value of
                slice island_id >0, then the pixel belongs to an island; otherwise, it belongs to the bkg.)
                        neighbors[slice_island_id[x,y,z],slice_island_id[x,y,z-1]]++; (a neighbor is found)
                end if
        end for
        for each slice island si2 on slice z do
                for each slice island si1 on slice z-1 do
                        if neighbors[si1,si2]>min(island_size[si1],island_size[si2])*beta then (beta=0.25)
                        (si1 and si2 are neighboring islands)
```

```
                        if (volume[si2]<0) then (island si2 does not belong to any volume)
                                volume[si2]=volume[si1] (si2 is now part of the volume of si1)
                                volume_size[volume[si1]]+=island_size[si2]
                                (add the size of island si2 to the size of the volume)
                        else
                                if (volume[sid1]!=volume[sid2]) then
                                (the neighboring islands are part of different volumes)
                                (the volumes are united)
                                        for each island si that is part of the same volume as si2
                                                volume[si]=volume[si1]
                                                volume_size[volume[si]]+=island_size[si]
                                        end for
                        end if
                end if
        end for
        if island si2 has no neighboring islands on slice z-1 then
                (create new volume id)
                volume[si2]=current_volume_id
                volume_size[volume[si2]]=island_size[si2]
                current_volume_id++
        end if
    end for
end for
Eliminate small volume islands - after this step, each volume island with volume_size>0 represents a
valid volume island
for each volume island vi do
    if volume_size[vi]<T then (threshold T is set to 10000)
        volume_size[vi]=0
    end if
end for
Stage 2. Hole filling step
for each slice z in the volume do
    do a BFS starting from the borders of the slice (connecting the background pixels)
    while there are still unvisited background pixels, do
        do a BFS starting with the first encountered background pixel
        (this BFS finds a background island surrounded by bone islands)
        check neighboring bone islands of current background island bi
        if there is only one neighboring bone island
            re-label all the pixels of bi as foreground
        end if
    end while
end for
End of pseudo code
```

4.3.2. Results

The bone reconstruction algorithm was tested on six noisy CT datasets representing body scans at the level of the hip. Each dataset contained slices where pixels belonging to different bones (the femur and the pelvis) were located in close proximity. Also, four datasets had a series of slices where a slice island belonging to the femur was located inside another island belonging to the pelvic bone, as illustrated in Fig. 4.12. Figure 4.13 shows a slice from each CT dataset.

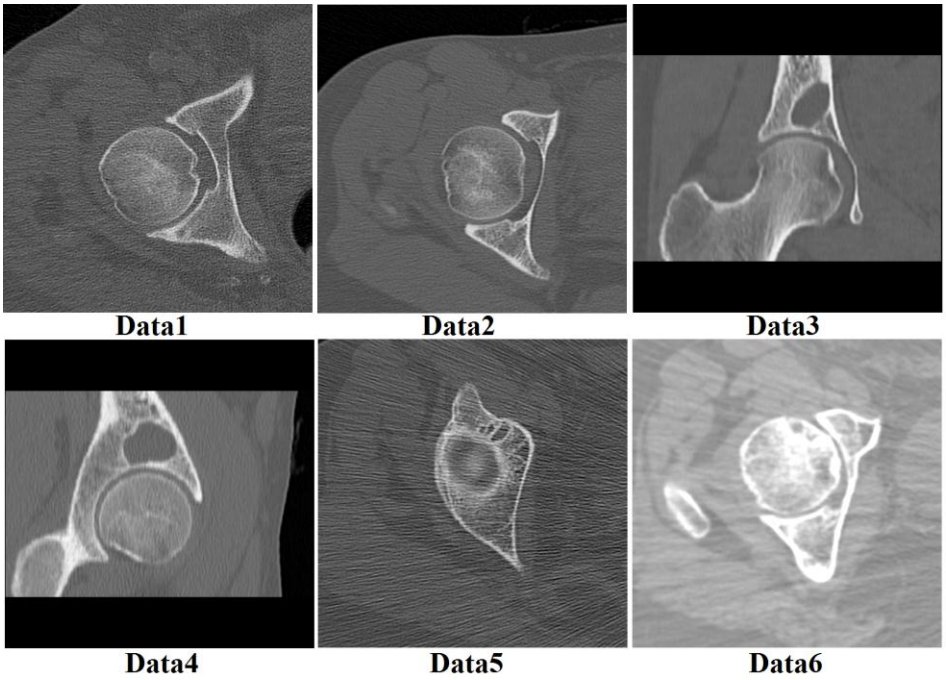

Fig. 4.13. Slices from the tested datasets

Due to the noise in the last two datasets Data5 and Data6, some user interaction was required. After the segmentation phase, a percentage of slices from these two datasets (approximately 8%) still contained pixels wrongly connecting different objects. These slices were manually modified, by removing the wrongly labeled foreground pixels. Table 4.6 shows the time required for the user interaction process. Table 4.6 also presents the computing times for the volume islands extraction and the hole filling step. The tests were made on an i7-720QM 1.60 GHz with 4GB DDR3 RAM.

The processing in the bone reconstruction stage is quite fast. The hole filling takes longer than the volume islands extraction because it performs several breadth first searches for each slice.

For the last two datasets that were very noisy, the most time consuming operation was the user interaction of adjusting the output generated in the bone segmentation phase. However, it must be mentioned that for the first four datasets there was no user interaction needed. The chosen segmentation method was the best as compared to the original active contours without edges algorithm and two segmentation algorithms based on graph cuts, regarding the time

consumed in the user interaction process. If there is previous knowledge regarding the number of bones scanned in a CT dataset and the method identifies the same number of bones, there is no user interaction required. On the contrary, if the number of bones automatically identified by the segmentation and reconstruction method is smaller than the actual number of bones, it means that the software segmentation wrongly connected different bones. In this case, the user has to correct the segmentation error.

Table 4.6. User interaction time for altering the output of the segmentation phase and computing times for the bone reconstruction process

Dataset (Size)	Data1 ($258x512^2$)	Data2 ($88x512^2$)	Data3 ($10x512^2$)	Data4 ($10x512^2$)	Data5 ($86x512^2$)	Data6 ($86x512^2$)
User interaction(sec)	-	-	-	-	300	360
Vol. islands extraction (sec)	1.61	0.57	0.08	0.08	0.58	0.57
Hole filling (sec)	82.99	28.93	3.08	3.02	27.94	28.16

After obtaining masks for all the volume islands where each one represents a different bone, the iso-surface describing the 3D model of the bones can be reconstructed with marching cubes. Fig. 4.14 presents the output of surface reconstruction from three of the datasets. After identifying different masks for the two bones, the user can decide which bone will be rendered. All three datasets contain parts of the femoral bone and the pelvic bone. In the left-side images, both masks are used, resulting in the rendering of both the femur and the pelvis. In the right-side images, only the mask of the femur is used.

Figure 4.15 presents the output of surface reconstruction performed on one of the datasets. In the left image, both masks are used. In the right image, only the mask of the pelvic bone is used.

4.4. CONCLUSION

This chapter approaches several methods that can improve the process of generating 3D models of personalized prostheses for hip arthroplasty. The first section proposes a computer assisted flow for producing the 3D model of a personalized prosthesis, based on the surface that approximates the bones of interest in hip arthroplasty (femoral and pelvic bones). The method does not guarantee in its current development stage that the produced prosthesis fits the femoral bone correctly. This is the reason we designed other techniques useful to produce personalized prostheses, which extract the volume occupied by particular bones, in our case the femoral bone. After the extraction of the volume occupied by the femur, the volume occupied by the femoral bone marrow can be extracted, and the 3D model of the prosthesis, which has to fit the interior of the bone, can be automatically generated.

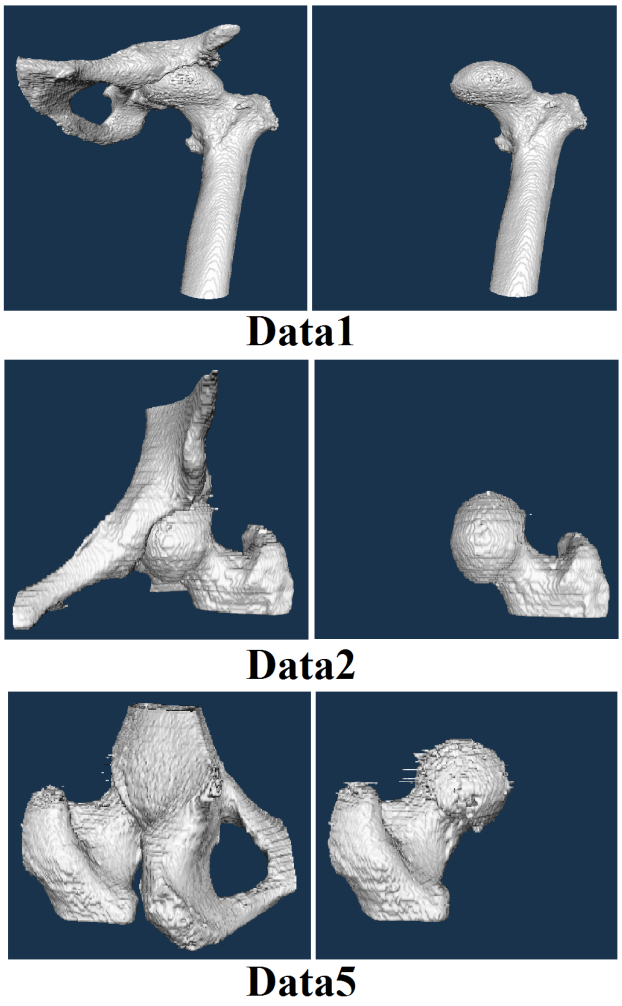

Fig. 4.14. Surface reconstruction with marching cubes from CT datasets, after applying the bone reconstruction algorithm: in the left images both the femur and the pelvis are rendered; in the right images only the femur is rendered

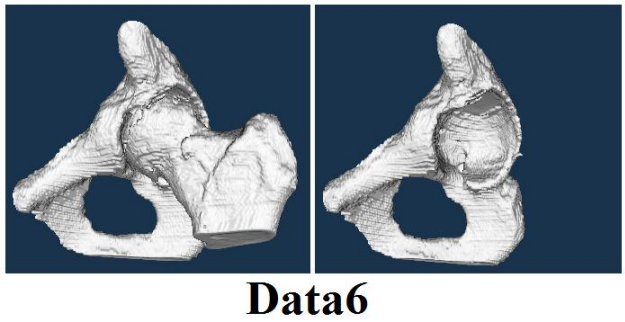

Fig. 4.15. Surface reconstruction with marching cubes from a CT dataset, after applying the bone reconstruction algorithm: in the left image both the femur and the pelvis are rendered; in the right image only the pelvic bone is rendered

The main contributions of the thesis included in this chapter are described below.

- Section 4.2 proposes a new 2D segmentation algorithm, based on active contours without edges, that discriminates quite well between bones and other tissues, but also between different bones like the femur and the pelvis. This method was tested on noisy CT datasets and compared to other well-known segmentation algorithms. The results show that the new segmentation algorithm is superior to the classic active contours algorithm without edges and other techniques based on graph cuts.

- Section 4.3 introduces a new step in the automatic/semi-automatic system that produces 3D models of customized artificial implants. The first stage of the method is the segmentation of the bones from CT datasets which is detailed in section 4.2. The segmentation method is slightly altered in order to provide better output for the next stage of the 3D segmentation (the bone reconstruction). The bone extraction process solves the problems of the hole filling step from the segmentation stage, by taking into account the relation between adjacent slices.

The next step in the generation of personalized prostheses could be the extraction of the interior of the femur, i.e., the volume occupied by bone marrow. After that, the skeletonization of the volume occupied by the femoral bone marrow could be performed. The skeletonization would help in computing a series of parameters. For example, if we have the skeleton of the femoral bone, then the position of the femoral body axis, the femoral neck axis and the diameter of the femoral body can be easily determined. These types of measurements are useful in the generation of customized artificial implants.

CHAPTER 5

EXTRACTION OF VESSEL CENTERLINES IN PERIPHERAL CT-ANGIOGRAPHY

The results of the research described in this section will be published in the paper:

G. Mistelbauer, A. Morar, A. Varchola, R. Schernthaner, Ivan Baclija, A. Köchl, A. Kanitsar, S. Bruckner, E. Gröller, "Human Reformation Maps – An Angiographic TreeMap Visualization using Double Curved Planar Reformations", to be submitted to the 6th IEEE Pacific Visualization Symposium, 2013.

A commonly used technique for vessel visualization is curved planar reformation (CPR), introduced by Kanitsar [35, 42, 43]. Another visualization method for CTA is the maximum intensity projection (MIP), after removing the bones from the dataset. We resume Kanitsar's description [43] of the two visualization methods.

Maximum intensity projection extracts high intensity structures from a dataset. For each image pixel, a viewing ray is cast into the volume. For each viewing ray, only the voxel with the maximum intensity is displayed. Therefore, bones will hide vessels if these structures cross each other. But, if the bones are previously segmented and masked, the maximum intensity projection method provides a very good overview of the vessel tree. The middle image from each of Figures 5.8-5.11 represents MIP views of CTA datasets after the removal of bones.

According to Kanitsar [35, 43], a curved plane is described by a curved line and a vector. For each point of the curved line, a straight line that is collinear to the vector is defined. Fig. 5.1 illustrates a curved line and the resulting curved plane that is built using a vector parallel to the X axis. Curved planar reformation is the process of extracting a set of vertices lying on the curved plane and displaying this set as a straightened plane.

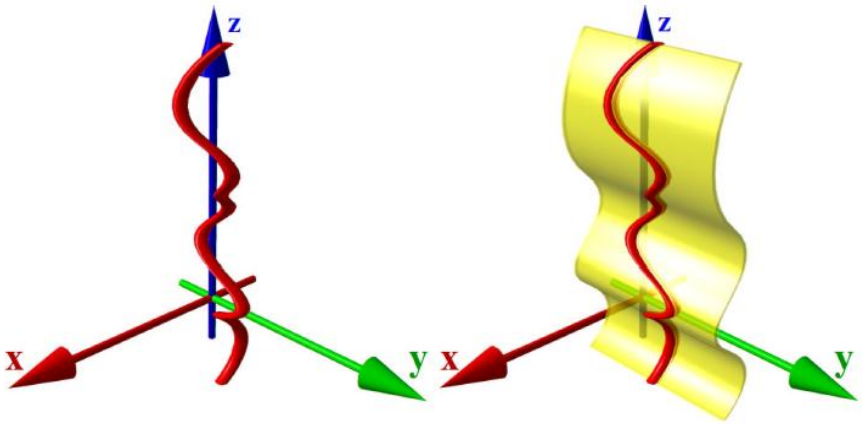

Fig. 5.1. A curved spatial line (left) and a curved plane defined by the curved line and a vector parallel to the X axis (source [35])

One of the requirements of exploring the vascular segments using CPR is to have a-priori knowledge about the structure of the vessels, especially their central axis. This chapter describes a method to detect the centerlines of the vessels. The method consists of three main stages that are depicted in Fig. 5.2: the segmentation of bone and vessel tissue from other tissues in the dataset, the vessel tracking based on user defined seed points and the centerline extraction. These stages are detailed in the following sections.

Fig. 5.3 shows the result of a multi-path CPR [48], a variant of the CPR that allows the visualization of the whole vascular tree. The graph that represents the vessel centerlines can be observed in a MIP view that displays both the bone and the vessel tissue.

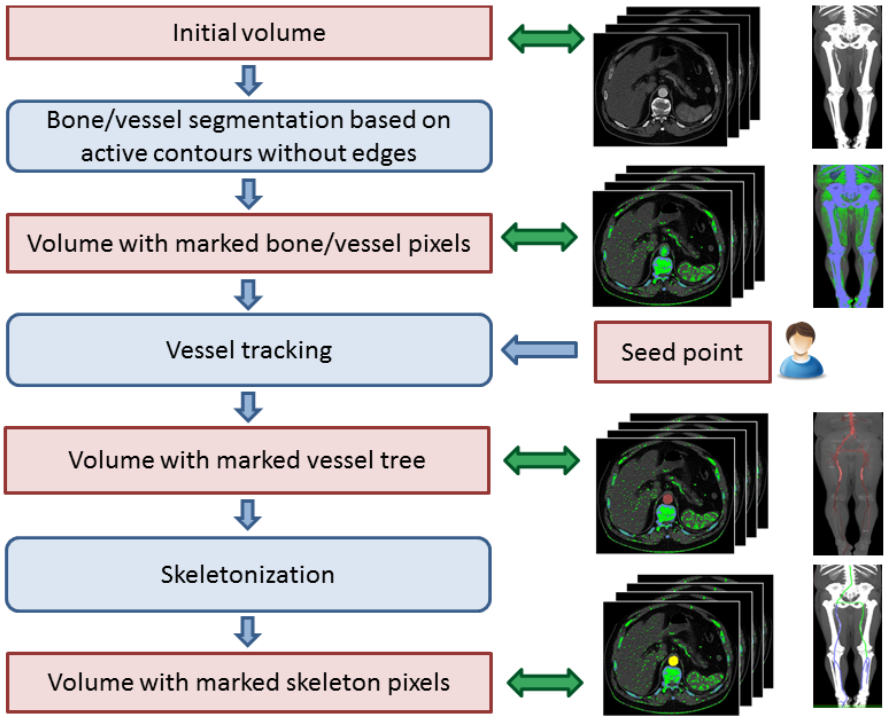

Fig. 5.2. Flow of the vessel centerline extraction algorithm

Fig. 5.3. (a) MIP view and (b) multi-path CPR for a CTA dataset

5.1. VESSEL/BONE SEGMENTATION

The high density objects in CTA datasets, i.e., the bones and the vessels, can be segmented from other tissues with a method similar to the one described in section 4.2. We make some adjustments to the slice based bone segmentation algorithm to extract both bones and vessels from conventional and dual-energy CTA. Other changes to the initial segmentation method are added to ease the process of differentiating between bones and vessels in the vessel tracking stage. The segmentation steps are illustrated in Fig. 5.4 and are slightly discussed below. More information on the reason behind introducing each step can be found in section 4.2.1. Based on the voltage applied (kV) in the data acquisition process, the steps of the segmentation differ slightly.

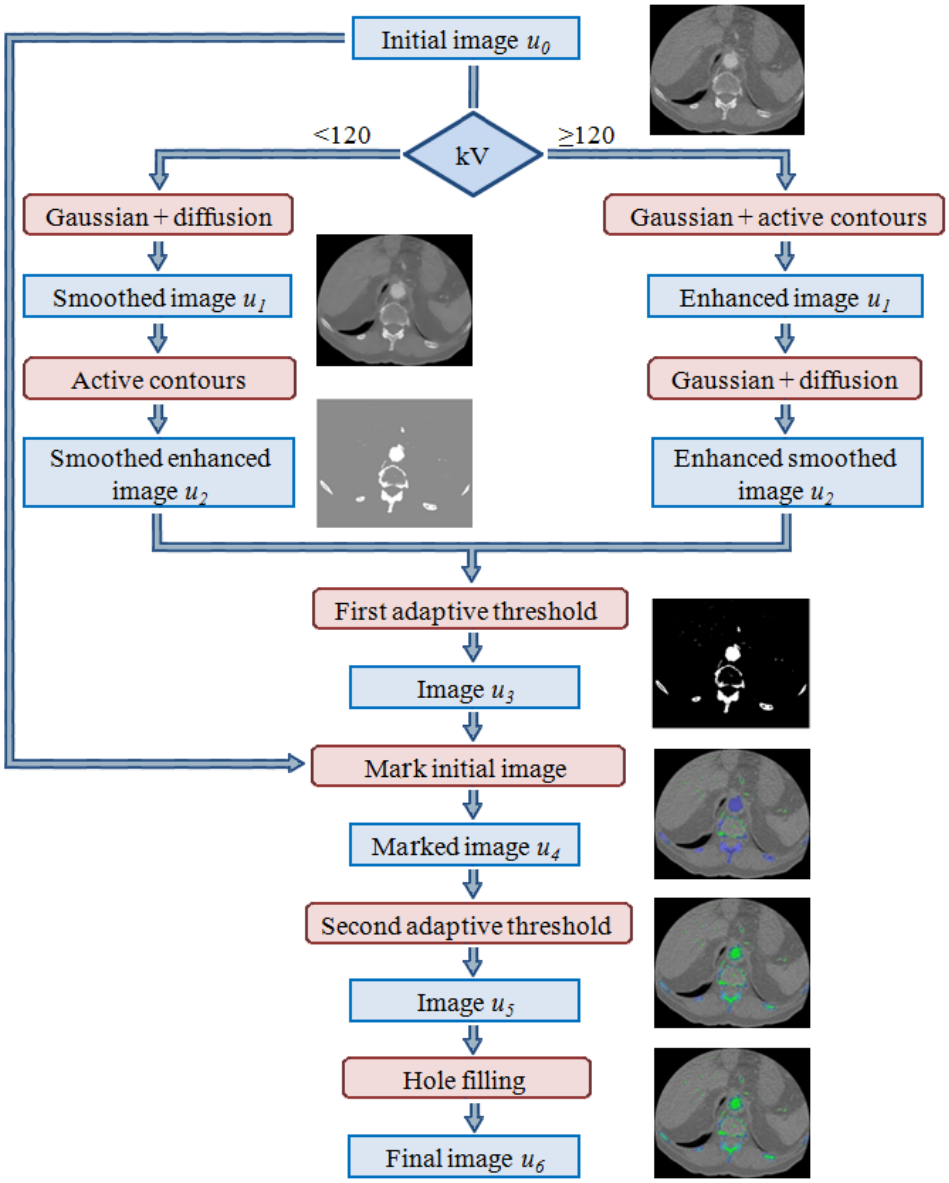

Fig. 5.4. Steps of the bone/vessel segmentation method

Step 1. Gaussian filter + nonlinear anisotropic diffusion / Gaussian filter + active contours

Lower energy CT images offer a better contrast between bone/vessel and other tissues, but the images have a granular aspect. This is why the slices from the lower energy datasets (less than 120 kV) are first smoothed. In order to preserve edges, we apply a nonlinear anisotropic diffusion filter.

The border between different objects in high energy CTs is already too thin, no smoothing being necessary in the beginning. This step is based on active contours without edges. Instead of segmenting the image into foreground and background, the step only enhances the contrast between different tissues in the slices. The processing of the first three steps is done on an auxiliary volume, so that the intensity values of the initial data are not distorted. Fig. 5.5(a) shows the output of the first step for a slice in a low energy CTA dataset.

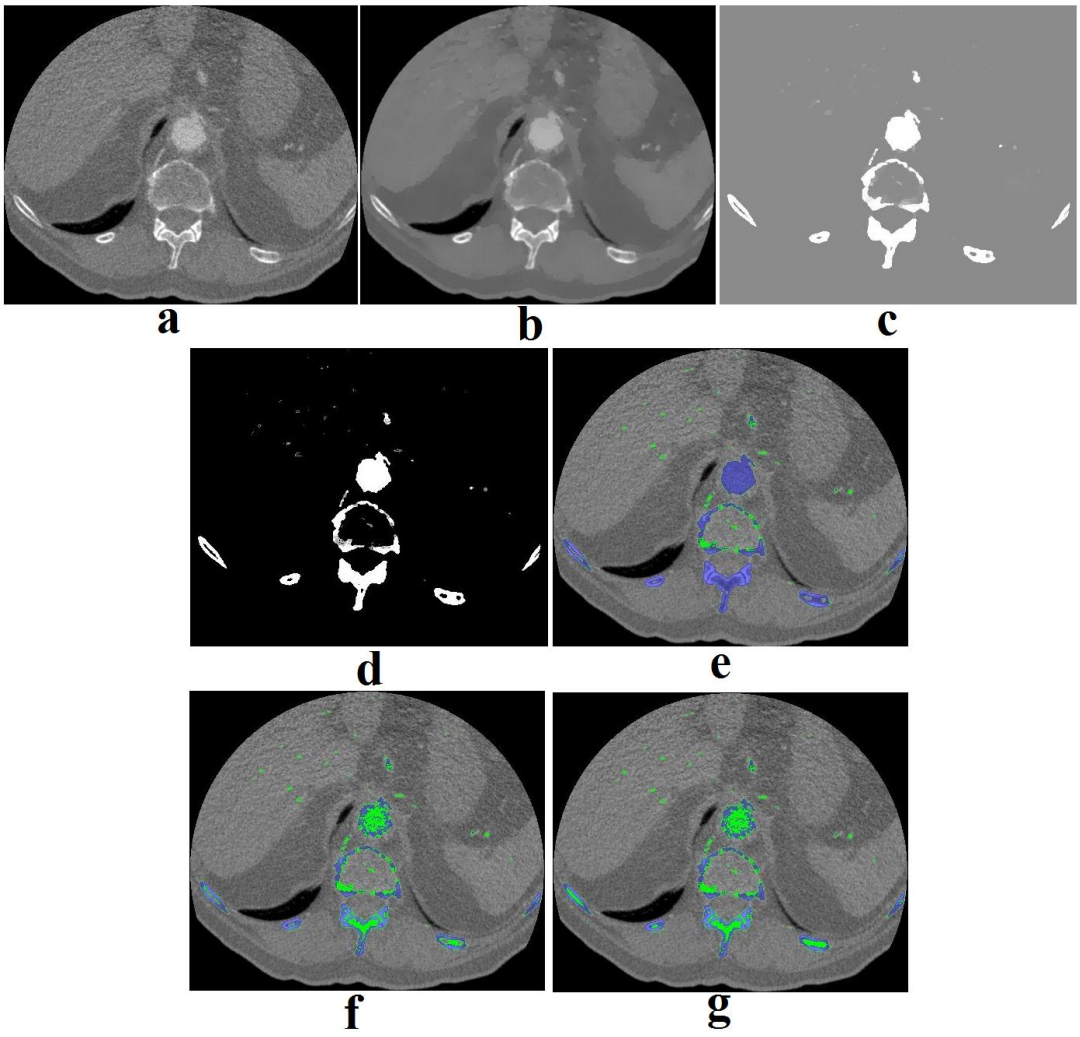

Fig. 5.5. Output images in the bone/vessel segmentation steps for a slice in a low energy CTA dataset: (a) original image, (b) image smoothed with nonlinear anisotropic diffusion (Step 1), (c) image enhanced with active contours without edges (Step 2), (d) result of first adaptive thresholding (Step 3), (e) output after marking different tissues on the original image (Step 4), (f) result of second adaptive threshold (Step 5), (g) final image after applying the hole filling step (Step 6)

Step 2. Active contours/ Gaussian filter + nonlinear anisotropic diffusion

The next step for the lower energy CTs is the contrast enhancement based on active contours without edges. For the high energy datasets we apply a Gaussian filter and a nonlinear anisotropic diffusion to remove small discontinuities within the object and the background. Fig. 5.5(b) depicts the result of applying the second step on a slice in a low energy CTA dataset.

Step 3.First adaptive thresholding

This step applies an adaptive thresholding to the pixels with the intensity between $I_{\max}/2-\Delta I_1$ and $I_{\max}/2+\Delta I_2$, where $I_{\max}$ is the maximum possible intensity while ΔI_1 and ΔI_2 are fixed parameters. All the pixels with intensity value smaller than $I_{\max}/2-\Delta I_1$ are considered background pixels and their intensity is set to 0. The pixels with intensity value greater than $I_{\max}/2+\Delta I_2$ are considered foreground pixels and their intensity is set to $I_{\max}$. The algorithm computes for each pixel the average M of the intensities from a neighborhood, based on expression (4.4). The pixels with intensity value less than M are considered background pixels.

Step 4. Marking different tissues in the initial image

Based on the intensity of the auxiliary volume, the pixels of the original dataset are divided into three categories and marked with different masks:

- strong pixels: all the pixels with intensity value equal to $I_{\max}$ in the auxiliary volume; in Fig. 5.5(e) we applied a blue mask on the strong pixels.

- candidate pixels: all the pixels with positive intensity value in the auxiliary volume; in Fig. 5.5(e) we applied a green mask on the candidate pixels.

- background pixels: all the pixels with zero intensity value in the auxiliary volume; in Fig. 5.5(e) we used a zero mask on the background pixels.

Step 5. Second adaptive thresholding

The output of the method is further refined based on the intensity values of the pixels from the original dataset. The second adaptive threshold is applied to all the pixels in the image, not only those belonging to the interval $\left[I_{\max}/2-\Delta I_1, I_{\max}/2+\Delta I_2\right]$. Unlike the first adaptive thresholding, the second one labels all the pixels with intensity value less than M as candidate pixels. In Fig. 5.5(f) it can be observed that a series of strong pixels are re-labeled as candidate pixels based on the intensity of the pixels in the original images.

Step 6. Hole filling

The island extraction step from the original bone segmentation algorithm is removed from the current method because it is time consuming and can also lead to the elimination of vessel tissue. Therefore, the next and last step is the breadth first search which finds the background pixels that are connected to the background pixels located at the image borders. All the background pixels that are not visited in the BFS are re-labeled as candidate pixels. Fig. 5.5(g) shows the output of the bone/vessel segmentation algorithm.

In the final image there are two types of candidate pixels:

- pixels that belong to bone/vessel tissue
- pixels that have been wrongly labeled because of the close proximity between certain parts of bones and vessels. This category can be detected and removed in the vessel tracking stage.

5.2. VESSEL TRACKING

The vessel tracking stage extracts the vessel tree based on a user-defined seed-point. The initial vessel tree contains not only healthy vessel tissue, but also calcifications, soft plaque or stents. The algorithm removes the calcifications/soft plaque based on the average intensity of the pixels inside the vessel tree. It also refines the result with a series of morphological operations so that the input of the next stage is not too noisy. Fig. 5.6 describes the steps of the vessel tracking algorithm.

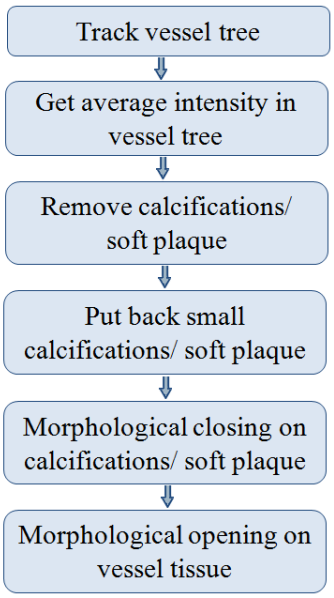

Fig. 5.6. Steps of the vessel tracking stage

Step 1. Initial vessel tree tracking

The user defines a seed-point from the strong/candidate pixels in a certain slice of the volume. The vessel tissue is then propagated to the neighboring strong/candidate pixels creating a vessel slice island, a concept similar to the one described in section 4.2.1. After the extraction of the current vessel slice island, its limits are set. The limits of a slice island are the x coordinate of the leftmost and rightmost pixels and the y coordinate of the topmost and the bottommost pixels in the island. The algorithm searches next for strong/candidate pixels in the adjacent slices of the current slice island, with the x and y coordinates between the limits of the current slice island. If such pixel exists, the island that is discovered with a BFS that starts from that pixel is investigated. If the new discovered island and the current vessel slice island have enough

neighboring pixels and do not vary too much in size, they are considered neighboring islands. In consequence, the new discovered island is also labeled as a vessel slice island.

Fig. 5.7 illustrates an example of slices where the variation in size between the islands suggests the existence of candidate pixels that have wrongly connected a bone and a vessel island. The algorithm investigates the current vessel island, i.e., the island with the pixels marked in red. On the adjacent slice, the neighboring island is divided into two smaller islands. One of them is connected to the current vessel island. The second island, that contains bone pixels, is not investigated further.

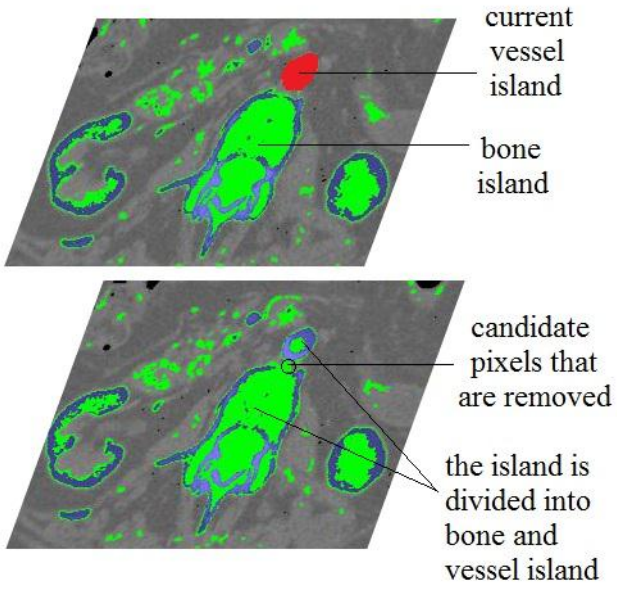

Fig. 5.7. Division of island into bone and vessel islands

The pseudo code of the initial vessel tracking algorithm is given below.

Pseudo code 5.1. Initial vessel tracking algorithm

```
get input from the user with seed-point (xs,ys,zs)
do BFS to obtain the vessel slice island vi that contains the seed-point (it propagates to all the strong and candidate pixels)
insert vessel island vi into queue q
while q is not empty, do
        Process current island ci
        get z coordinate of current island ci
        Search for neighboring islands on slice z-1
        find_neighboring_island(ci,z-1)
        Search for neighboring islands on slice z+1
        find_neighboring_island(ci,z+1)
end while
End of pseudo code
```

The next pseudo code describes the propagation of the current vessel island to the neighboring vessel islands.

Pseudo code 5.2. Propagation of vessel tissue to the neighboring islands

```
find_neighboring_island(i,zn)
for every pixel p(x,y,z) in the current vessel island i do
        if pixel pn(x,y,zn) has not been visited then
                detect island in that contains pixel pn
                compute size of island in
                compute number of neighboring pixels np between islands i and in
                if np>min(size(i),size(in))*alpha, then (alpha is a constant)
                        If the two islands are neighbors, then
                                Check if there is a big difference in size
                                if size(in)/size(i) >beta then (beta is a constant)
                                        Island in has to be divided into bone and vessel islands
                                        Remove candidate pixels that are not connected to vessel
                                                                        islands on the adjacent slices
                                        re-segment island in
                                else
                                        add island in to the queue of vessel islands q
                                end if
                        end if
        end if
end for
if there is a neighboring island that has been re-segmented then
        Repeat the whole algorithm
        find_neighboring_island  (i,z)
end if
End of pseudo code
```

Step 2. Compute average intensity within the vessel tree

The second step computes the average intensity I_{avg} of the previously extracted pixels, i.e., the pixels of the vessel tree. The average intensity is necessary for the next step that removes the calcifications and the soft plaque from the vessel tree.

Step 3. Remove calcifications/soft plaque

The calcifications/stents in CTA slices have a bigger density than the healthy vessel tissue. On the contrary, the density of the soft plaque is lower than the density of vessel tissue. This step removes from the vessel tree all the pixels with the intensity value less than $I_{avg} - \Delta I_3$ or greater than $I_{avg} + \Delta I_4$, where ΔI_3 and ΔI_4 are fixed parameters.

Step 4. Put back small calcifications/soft plaque

The result of the vessel tracking algorithm is used for the computing of the vessel centerlines based on skeletonization. Therefore, for a correct extraction of the skeleton, the vessel tree should have a smooth aspect, without calcifications/soft plaque in the interior of the vessel. This step checks the size of the islands previously removed from the vessel tree based on the average intensity of the vessel pixels. If the determined size is less than a given threshold, we assume that there is no calcification or soft-plaque at that location.

Step 5. Closing on calcifications/soft plaque

This step performs a morphological closing on the pixels that belong to calcifications/soft plaque tissue. It first dilates the unhealthy vessel tissue and then erodes the pixels that are neighbors with a healthy vessel pixel.

The pseudo code of the closing operation on calcifications/soft plaque is given below.

Pseudo code 5.3. Morphological closing on calcifications/soft plaque for one slice

```
for each pair of coordinates (x,y), do
        initialize dilateArray      (used for morphological dilation)
        initialize erodeArray       (used for morphological erosion)
end for
for each vessel island vi in the slice do
        radius=sqrt(island_size/alpha) (island_size is the size of the island, alpha is a constant)
        (the morphological element is a circle of radius radius)
        Dilate non-vessel
        for each pixel p in vi do
                if p is not vessel pixel then
                        dilateArray[p]=1
                        for y=p.y-radius to p.x+radius do
                                for x=p.x-radius to p.x+radius do
                                        if (p.x-x)*(p.x-x)+(p.y-y)*(p.y-y)<=radius*radius do
                                                if pixel pn(x,y) is vessel pixel then
                                                        dilateArray[p]=1
                                                end if
                                        end if
                                end for
                        end for
                end if
        end for
        Erode non-vessel
        for each pixel p in vi do
                if dilateArray[p]==1 then
                        erode=false;
                        for y=p.y-radius to p.y+radius do
                                for x=p.x-radius to p.x+radius do
```

```
                                        if (p.x-x)*(p.x-x)+(p.y-y)*(p.y-y)<=radius then
                                                if dilateArray[pn(x,y)]=0 and pn is vessel island then
                                                        erode=true
                                                        break
                                                end if
                                        end if
                                end for
                                break if erode=true …
                        end for
                        if erode=true then
                                erodeArray[p]=1
                        end if
                end if
        end for
        for each pixel p in vi do
                if dilateArray[p]=1 then
                        p is non-vessel island
                if erodeArray[p]=1 then
                        p is vessel island
        end for
end for
End of pseudo code
```

Step 6. Opening on vessel tissue

This step erodes all the vessel pixels that are not completely surrounded by healthy vessel pixels and then dilates the vessel pixels. In the dilation process, calcifications and soft plaque pixels can be re-labeled as vessel pixels, but the background pixels remain unchanged.

The result of the vessel tracking step is a volume where the following masks are applied:

- blue/green mask on bone pixels
- red mask on vessel pixels
- pink mask on calcifications/soft plaque/stents

If the vessel tree has not been completely extracted in an iteration, the vessel tree tracking step can be repeated with other user-defined seed-points.

5.3. CENTERLINE EXTRACTION

The centerlines of the vessel trees are extracted with a skeletonization technique that is applied to all the vessel pixels. The skeleton is built using the 3D thinning algorithm proposed by Lee et al. [36].

The original algorithm performs the skeletonization on a 3D binary array where the points of interest have the value 1 and the background points, the value 0. Our implementation reduces the computing times by shrinking the search space of the foreground points. The vessel tracking stage produces an array that stores information for each extracted vessel island. Among the stored information, the array saves the limits of each vessel island, i.e., the maximum and minimum values of the x and y coordinates of the pixels within that island, $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$. Therefore, the algorithm does not investigate all the pixels in the volume, but only those pixels within the limits of the vessel islands, which represent approximately 3% of the total number of pixels.

The skeleton that was obtained in the 3D binary array is then converted to a graph that is further used for the investigation of the vessel tree with the CPR method.

5.4. RESULTS

The proposed algorithm has been tested on conventional CTAs, as well as on low and high energy CTAs. The bone/vessel segmentation stage is implemented in CUDA, in a similar manner as to the one described in section 4.2.2. The other steps, the vessel tracking, the skeletonization and the conversion of the skeleton into the graph are implemented on the CPU.

Table 5.1 shows the computing times for the processing of some of the CTA datasets. We chose datasets with different energy levels and of various sizes.

Table 5.1. Computing times for the steps of the new algorithm that extracts the centerlines of the vessels

Dataset size (pixels)	Energy level (kV)	Bone/vessel segmentation (sec)	Vessel tracking (sec)	Centerline extraction	
				Skeletonization (sec)	Graph construction (sec)
256×512^2	120	12.13	1.9	1.68	6.78
767×512^2	140	41.13	1.28+1.39+1.33+1.43	1.41	2.99
1152×512^2	120	56.71	4.77	3.21	8.38
1305×512^2	120	59.78	3.21+2.92	3.15	26.04
1800×512^2	80	68.72	7.94	4.61	6.29

The algorithm is quite fast and requires very little human interaction for defining the seed point(s). The vessel trees in the second and the forth dataset are tracked based on multiple seed points. The vessel tracking column from Table 5.1 presents the computing times for the extraction of each sub-vessel tree.

Four of the datasets are visualized in Figures 5.8-5.11 with the MIP (maximum intensity projection) method before and after applying the vessel tracking step. The graphs that approximate the centerlines of the vessels are also shown in these figures.

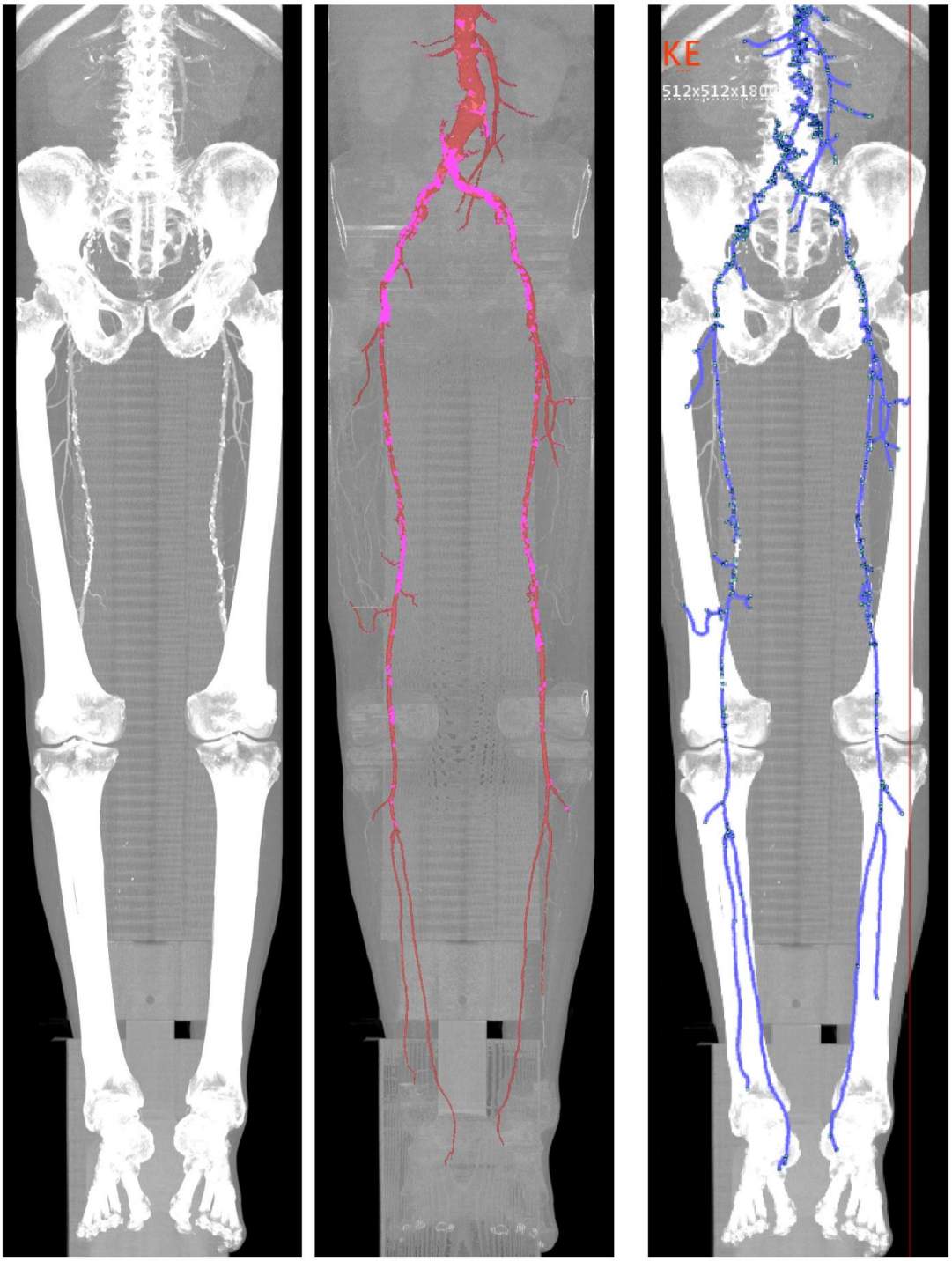

Fig. 5.8. MIP view of an 80 kV dataset of size $1800x512^2$: initial volume (first image), volume with marked vessels and calcifications/soft plaque (second image), view of the initial volume and the graph that approximates the centerlines of the vessels (third image)

The artificial implants can become an impediment for the correct extraction of bone and vessel tissue. The metals from the implants introduce artifacts in the CTA images that affect the attenuation of vessels. Such an example is given in Fig. 5.9 where an artificial implant in the left part of the body influences the vessel tree tracking process. At the level of the prosthesis head, the vessel tissue could not be segmented from other tissues. Even though multiple seed-points were defined, the vessel tree still had some discontinuities in the area of the artificial implant.

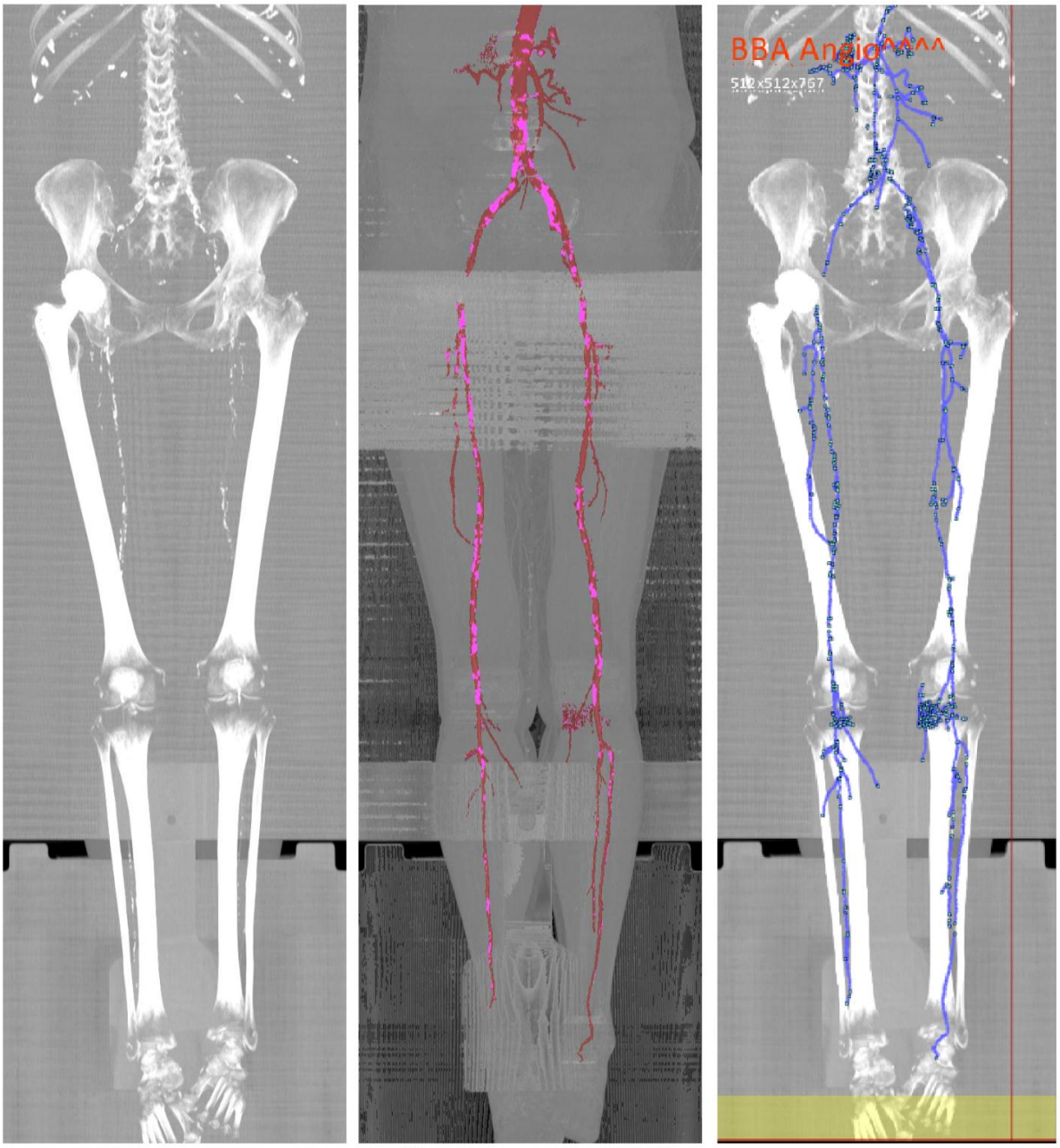

Fig. 5.9. MIP view of an 140 kV dataset of size $767x512^2$: initial volume (first image), volume with marked vessels and calcifications/soft plaque (second image), view of the initial volume and the graph that approximates the centerlines of the vessels (third image)

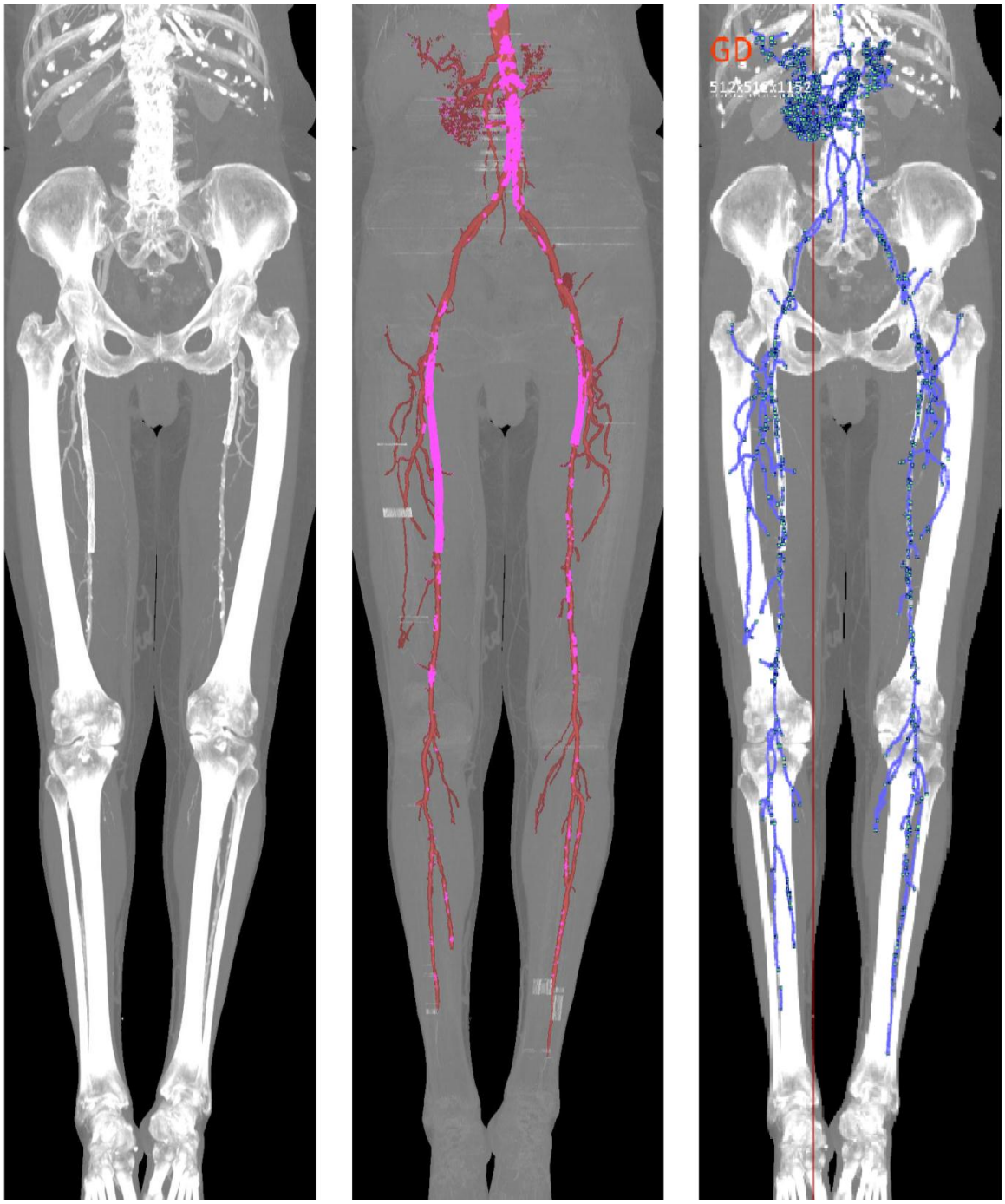

Fig. 5.10. MIP view of an 120 kV dataset of size $1152x512^2$: initial volume (first image), volume with marked vessels and calcifications/soft plaque (second image), view of the initial volume and the graph that approximates the centerlines of the vessels (third image)

Fig. 5.11 shows the view of a dataset belonging to a patient who suffered a surgical intervention – the insertion of a vascular bypass. In the vessel tracking process, the islands that belong to the bypass differ considerably from the vessel islands in the adjacent slices, because of the horizontal positioning of the bypass. Therefore, for the extraction of the whole vessel tree, two seed points are needed: one at the top slices of the dataset and one at the level of the bypass.

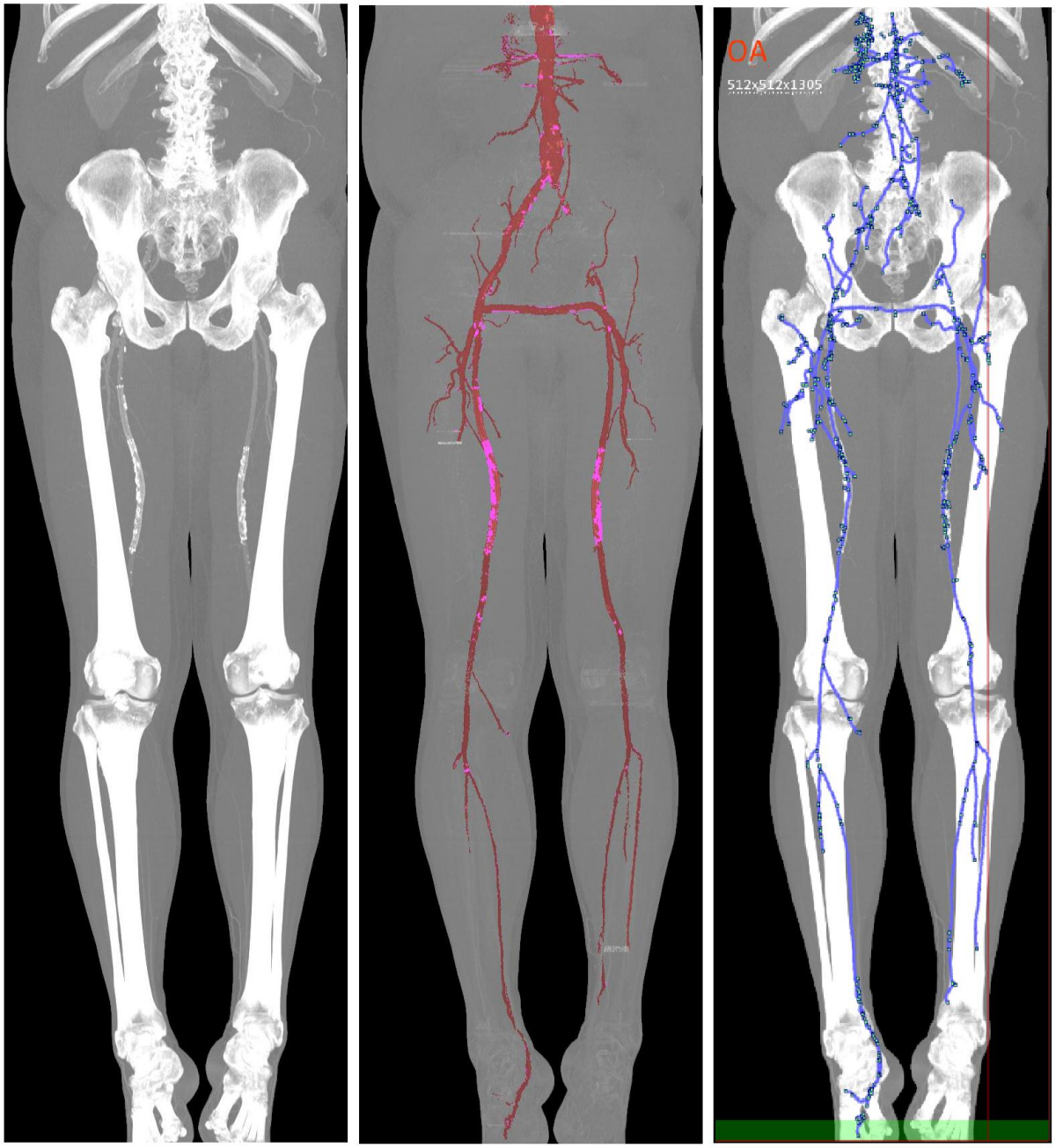

Fig. 5.11. MIP view of an 120 kV dataset of size 1300x512^2: initial volume (first image), volume with marked vessels and calcifications/soft plaque (second image), view of the initial volume and the graph that approximates the centerlines of the vessels (third image)

The centerline extraction algorithm has been successfully integrated into the AngioVis application [40] that is aimed on clinical processing and visualization of peripheral CTA datasets. The method was presented to radiologists from the Medical University of Vienna, Department of Cardiovascular and Interventional Radiology. In their opinion, the segmentation, the semi-automatic vessel tracking and the centerline extraction greatly reduces the time spent for surgical planning. Their suggestion is to add a series of tools that can adjust the results of the centerline extraction in case of errors or in case of incomplete tracking of the vessel tree.

5.5. CONCLUSIONS

This chapter introduces a series of methods that enable the processing of conventional and dual-energy CTA datasets for the purpose of investigating the vessels. Curved planar reformation, the state of the art in vessel visualization, requires the previous extraction of the vessel centerlines. The main contribution of this chapter consists of a method to obtain the vessel centerlines in three steps:

- The bone/vessel extraction step, which is based on the active contours model without edges. It represents a variant of the bone segmentation method that is described in chapter 4. The changes made to the initial algorithm ensure the possibility to handle different medical imaging modalities like conventional or low/high energy CTAs. The vessel/bone pixels are divided into two categories, strong and candidate pixels, and are further used in the vessel tracking process.
- The vessel tracking algorithm that starts from a user-defined seed-point and propagates the vessel tissue to its neighboring pixels, detecting vessel slice islands. The vessel tissue is then propagated to neighboring slice islands from adjacent slices. The propagation stops when there are big variations in the size of the investigated islands. This rule assures that the bone islands, which usually occupy a larger area than the vessel islands, are not incorrectly labeled as being part of the vessel tree. After the initial vessel tracking step, the calcifications and soft plaque are removed from the vascular tree. The skeletonization stage is very sensitive to noise. Therefore, in order to obtain a smooth skeleton that approximates the vessel centerlines, the vessel tissue is refined with two morphological operations: a closing on the non-vessel tissue and an opening on the vessel tissue.
- The central axes of the vessel arteries are represented in a graph that is obtained from the skeleton of the vessel tree

The centerline extraction method is fast and requires very little user-interaction. However, artifacts and noise introduced by the image acquisition device or by metals inside the scanned body can become a drawback for the correct segmentation of bone/vessel tissue. This drawback can be overcome with a series of user interaction tools that can adjust the results of the centerline extraction algorithm. The application should give the user the possibility to change the position of certain points of the graph that approximates the centerlines. The user should also be

able to connect vessels that were incompletely extracted and to reconstruct the missing paths. The method described in this chapter has been implemented and integrated in the AngioVis application [40]. The work presented in this chapter is part of the Knowledge Assisted Sparse Interaction for Peripheral CT-Angiography (KASI) project, supported by the Austrian Science Fund (FWF) grant no. TRP 67-N23. The datasets are courtesy of the Kaiser-Franz-Josef Hospital and the General Hospital of Vienna.

CONCLUSIONS

1. ORIGINAL CONTRIBUTIONS

This thesis describes the results of our research in the fields of hip arthroplasty and angiology using image processing and visualization techniques. We studied and improved some existing algorithms and we designed new algorithms and methods of medical data processing and visualization.

The main contributions of the thesis are summarized below.

- Chapter 2 presents a practical contribution: an application which can automatically detect the parameters of interest in hip arthroplasty. We observed that some salient parts of the bone contour can be approximated by simple curves like lines and circles. The Hough transforms were chosen to approximate these imperfect instances of lines and circles in the radiographic images.

  The input of the Hough transform is the matrix of gradient magnitudes. This matrix can be obtained with gradient approximation masks, like Sobel, Roberts or Prewitt operators. But, for thinner borders between bones and other tissues and for less noise pixels in the input of the Hough transforms, the Canny edge detector is a better alternative to the gradient approximation operators.

  The main steps of the method we used in our experiments and described in this thesis are:
  - the extraction of the contour in radiographic images with Canny edge detector
  - the detection of salient parameters in radiographic images that can be approximated by lines and circles with variants of the Hough transform for circles and the Hough transform for lines
  - the extraction of the other parameters that are important for hip arthroplasty based on the previously extracted ones

  The application is pretty accurate regarding the correct extraction of the parameters. The medical experts that evaluated the method gave a positive feedback, stating that the parameter extraction algorithm considerably reduced the time spent for the preparation of surgeries. Also, the results of the automatic detection method can be modified by human intervention in case of errors.

- The feature extraction algorithms that were used in the previously mentioned application, the Canny edge detector and the Hough transforms, are time consuming for large images. The second contribution of chapter 2 consists of a series of improvements to the existing parallel implementations of the Canny and Hough algorithms. We investigated several approaches that reduce the computing times, based on the particularities of the radiographic images and the characteristics of the CUDA architecture.
    - We designed and tested multi-GPGPU implementations of the Canny edge detector and the Hough transform for lines. The results prove the scalability of the multi-GPGPU approach.
    - Other experiments were focused on reducing the overhead of the transfers between the CPU and the GPU with the use of the page locked memory. These implementations add a performance gain, but are limited by the size of the pinned memory, which is far smaller than the pageable one.
    - The last approach that improves the computing times of the Hough algorithms is based on the reduction of the search spaces. The number of possible values for the parameters of the Hough algorithms is decreased based on the particularities of the investigated radiographic images and the characteristics of the structures that need to be extracted.

The results of the research described in chapter 2 prove the efficiency of the parallel implementations of image processing algorithms using the GPGPU technology.

Chapter 3 introduces two contributions in the field of volume rendering.

- The first main contribution of this chapter is an improvement to the CUDA implementation of the marching cubes algorithm. The classic GPU marching cubes from the CUDA SDK Example [29] is a fast alternative to the CPU volume rendering. However, this implementation can handle datasets of relatively small sizes (up to 256^3). We proposed a new approach to the CUDA marching cubes algorithm that splits the initial volume into sub-volumes, which are processed serially on the GPU. This implementation is slightly slower than the classic GPU one, but can handle datasets up to 32 times larger.
- The second contribution of chapter 3 consists of three silhouette rendering methods that are suited for different characteristics of datasets and GPU cards:
    - The first method enables the silhouette rendering based on the previously extracted iso-surface with marching cubes. The iso-surface is sent to a vertex shader and to a geometry shader that emits the lines of the silhouette by connecting the points where the normal is perpendicular to the view vector. This method is suited for GPU cards with enough memory to store the dataset, the iso-surface and the silhouette at the same time. This requirement considerably limits the size of the datasets to be handled. The main advantage of this approach is the speed of the rendering process due to the fact that the position shifting of the viewer does not affect the reconstruction of the iso-surface.
    - The second method is suited for GPUs that have enough memory to store only the dataset and the silhouette at the same time. The new silhouette rendering method is

composed of two steps: the silhouette reconstruction and a ray based visibility test. For each end-point of a silhouette segment, a ray is cast from that point to the eye, intersecting the volume. If the ray intersects the iso-surface, then the current end-point is occluded. This method requires less GPU memory than the first algorithm, but the ray based visibility test is time consuming.

- The third method is suited for GPUs that have enough memory to store only the dataset. The depth information of the triangles of the iso-surface and of the silhouette segments is updated into two depth buffers through a CUDA rasterizer. The depth buffers are then combined for obtaining the final color buffer that is displayed on the screen. The CUDA rasterizer is an alternative to the GPU memory limitation, but does not offer interactive frame rates, because it is not specialized on rendering like the hardware rendering mechanisms.

All the three proposed methods can be designed to handle larger datasets by splitting the initial volume into sub-volumes that are processed serially on the GPU.

Chapter 4 describes our contributions in the field of image segmentation. The first section of the chapter introduces several ideas for the generation of 3D models of prosthesis for hip arthroplasty, based on the output of the marching cubes algorithm. These ideas have not been completely implemented and tested and have to be further refined in order to be used in practice. The main drawback of the marching cubes algorithm, relative to the fitting of a prosthesis in the femoral bone, is the impossibility to extract different iso-surfaces, each representing an individual bone, based on an iso-value. A method that discriminates between different bones could be a step in the automatic generation of 3D models of prostheses. If the volume occupied by the femoral head is detected, then this volume can be processed for the purpose of extracting different parameters of interest for hip arthroplasty. After the extraction of these parameters, the 3D model of the prosthesis can be generated.

- The first contribution of this chapter is a 2D segmentation method based on active contours without edges. The method consists of several steps that are slightly described.
  - The first step enhances the contrast between the foreground and the background with a modified implementation of the active contours model without edges
  - The second step removes small discontinuities within the foreground, especially near the frontiers, by applying a nonlinear anisotropic diffusion filter
  - The third and the forth step apply adaptive thresholdings, one on the output of the second step and the other one on the initial image.
  - The fifth step combines the outputs of the two adaptive thresholdings. The output of this step consists of an image with clear frontiers between different objects
  - The sixth step extracts the foreground slice islands and removes those islands that are composed of pixels with low intensity
  - The last step re-labels the background pixels that are surrounded by foreground slice islands.

Our method was compared with three existing segmentation techniques. The accuracy regarding the correct labeling of background and foreground pixels was comparable or even

better than that of the other segmentation methods. However, the difference that proved the superiority of our method is the accuracy regarding the discrimination between different objects that are positioned in close proximity of each other.

The active contours and the nonlinear anisotropic diffusion steps are time consuming because they require a large number of iterations until reaching a final state. This drawback was overcome with the use of GPU programming. All the steps of the algorithms except the island extraction are executed in parallel, being implemented with CUDA. Our implementation proved that the GPGPU technology adds a significantly performance gain to the segmentation of large datasets.

- The second contribution described in chapter 4 is a 3D bone segmentation method that can be used for extracting the volume occupied by the femoral bone in CT datasets. The foreground slice islands obtained with the previously described 2D segmentation method based on active contours without edges are connected in order to construct foreground volume islands. Each volume island represents a different bone in the dataset. Our new method can extract the geometric representation of a particular bone from the dataset, while the output of the marching cubes algorithm is an iso-surface containing all the bones from the dataset. This is an advantage of our method, because automatic measurements and parameter extraction can be accomplished for the particular bone of interest.

Chapter 5 presents the results of our research in the field of angiology. One of the most commonly used techniques for vessel visualization is curved planar reformation. The input of this visualization method consists of the central axes of the vessels. The main contribution described in this chapter consists of a method that extracts the vessel centerlines with very little human interaction.

- The first step of this method is a 2D segmentation method that is based on the segmentation method described in chapter 4. It differentiates between bone/vessel and other tissues in conventional and dual energy CTAs. The output of the segmentation is the set of slices where the bone/vessel pixels are marked as strong or candidate pixels. The strong pixels belong to the bone/vessel pixels with high certainty. The candidate pixels could be part of the bones or the vessels, or could be pixels that wrongly connect bones and vessels.
- The next step in the algorithm is the initial vessel tree tracking that starts with a user-defined seed point. The vessel tissue propagates from the current pixel to its neighboring strong/candidate pixels and from vessel slice islands. The neighboring islands of the current vessel slice island are then investigated and re-labeled as vessel islands if they have common characteristics with the current vessel island. The propagation stops if there is a big variation in the size of the islands. If the vessel tree is incomplete, the tracking step can be repeated with other user-defined seed points.
- The extracted vessel tree contains both healthy tissue and calcifications/soft plaque. The vessel tree is next refined with the removal of non-vessel tissue, which has a very high or very low intensity as compared to the average intensity within the vessel tree. Two morphological operations, a closing on the calcifications/soft plaque and an opening on the vessel tissue ensure a smooth shape of the vessel tree.

- The centerlines are extracted with a skeletonization method based on 3D thinning.

  The obtained centerlines can be used for the investigation of the vessel tree in CTA datasets with different variants of the curved planar reformation method.

2. FUTURE RESEARCH

This thesis describes several contributions to the analysis and visualization of data from medical images. Although this is a very discussed field, there are still a lot of research directions that can be followed. The characteristics of the medical images, the poor contrast between different tissues, the noise introduced during the medical image acquisition process and the inhomogeneity in intensity within the same object represent a drawback for automatic image analysis applications. These medical image investigation methods fail in many cases to produce similar or better results than those obtained by domain experts. This is a reason why the field of analysis and visualization of medical data is still open to research.

One of our future research directions is the design of a fully automatic method to produce 3D models of customized prostheses in arthroplasty (not only at the level of the hip). This would significantly reduce the time spent for choosing the prostheses. It would also improve the medical process, since the prostheses would perfectly fit the physiological characteristics of the patients.

Another direction of research is the vessel investigation with variants of the curved planar reformation method that requires other parameters besides the vessel centerlines. Such a method is based on the estimation of the minimum and maximum diameters of the vessels, as well as the orientation of the vectors with the minimum and maximum length. Also, the segmentation of the vessels/bones can be further improved, especially for high energy CTA datasets, where the contrast between vessels and other tissues is low.

The research described in this thesis can also be continued with other GPGPU implementations of analysis and visualization methods, in order to handle larger and larger data volumes and to run faster, preferably in real time.

LIST OF PUBLICATIONS AND PROJECTS

Publications connected to this thesis:

- **Journals**:

***A. Morar**, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner*, "Medical Image Processing in Hip Arthroplasty", WSEAS TRANSACTIONS on SIGNAL PROCESSING, Vol. 6, Issue 4, pp. 165-174, 2010.

***A. Morar**, F. Moldoveanu, V. Asavei, L. Petrescu, A. Moldoveanu, A. Egner*, "GPGPU Based Non-photorealistic Rendering of Volume Data", accepted for publication in the Journal of Control Engineering and Applied Informatics – to appear in no.3, 2012 (ISI).

***A. Morar**, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner*, "Multi-GPGPU Based Medical Image Processing in Hip Replacement", accepted for publication in the Journal of Control Engineering and Applied Informatics – to appear in 2013 (ISI).

***A. Morar**, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner*, "CT Image Processing in Hip Arthroplasty", accepted for publication in The Scientific Bulletin of University POLITEHNICA of Bucharest, Series C Electrical Engineering and Computer Science -to appear in no.1, 2013.

- **Conference Proceedings:**

***A. Morar**, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner*, "Computer Assisted Analysis of Orthopedic Radiographic Images", Proceedings of the 9th WSEAS International Conference on SIGNAL PROCESSING, pp. 66-71, Catania, 2010 (ISI).

*V. Asavei, A. Moldoveanu, F. Moldoveanu, **A. Morar**, A. Egner*, "GPGPU for Cheaper 3D MMO Servers", Proceedings of the 9th WSEAS International Conference on TELECOMMUNICATIONS and INFORMATICS, pp. 238-243, Catania, 2010 (ISI).

***A. Morar**, F. Moldoveanu, A. Moldoveanu, V. Asavei, C. Boiangiu, A. Egner*, "Computer Assisted Analysis of 2D/3D Medical Images", Proceedings of the 21st International

DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 1273-1274, Zadar, 2010 (ISI).

V. Asavei, A. Moldoveanu, F. Moldoveanu, C. Boiangiu, ***A. Morar****, A. Egner,* "Innovative 3D MMO Servers Architectures Based on GPGPU", Proceedings of the 21st International DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 649-650, Zadar, 2010 (ISI).

L. Petrescu, ***A. Morar****, F. Moldoveanu, V. Asavei,* "Real Time Reconstruction of Volumes from Very Large Datasets using CUDA", Proceedings of the 15th International Conference on System Theory, Control and Computing, pp. 462-466, Sinaia, 2011 (IEEE).

A. Morar*, F. Moldoveanu, A. Moldoveanu, V. Asavei, A. Egner,* "Computer Assisted Insertion of Prostheses Based on Medical Images", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.636-641, Bucharest, 2011.

V. Asavei, F. Moldoveanu, A. Moldoveanu, A. Egner, ***A. Morar****,* "Multi GPGPU Optimizations for 3D MMO Virtual Spaces", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.642-646, Bucharest, 2011.

A. Morar*, F. Moldoveanu, E. Gröller*, "Image Segmentation Based on Active Contours without Edges", Proceedings of the 8th International Conference on Intelligent Computer Communication and Processing, Cluj, 2012 (IEEE).

- **Posters:**

A. Morar*, F. Moldoveanu, V. Asavei,* "Computer Assisted Analysis of Medical Images", poster presented at CATIIS PhD Student's Day, Bucharest, 2010.

Publications that were not related to the research described in this thesis:

- **Conference Proceedings:**

C. Boiangiu, D. Rosner, A. Olteanu, ***A. Morar***, "Automatic Slanted Edge Target Validation in Large Scale Digitization Projects", Proceedings of the 21st International DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 131-132, Zadar, 2010 (ISI).

A. Egner, F. Moldoveanu, A. Moldoveanu, V. Asavei, ***A. Morar****, C. Boiangiu,* "Testing the Interoperability of HL7-based Applications Using TTCN-3", Proceedings of the 21st International DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 1279-1280, Zadar, 2010 (ISI).

A. Moldoveanu, V. Asavei, F. Moldoveanu, ***A. Morar****, A. Egner, C. Boiangiu,* "Turning Nearshoring into a Success Managing Technical Background Differences", Proceedings

of the 21st International DAAAM Symposium/ 4th European DAAAM International Young Researchers and Scientists Conference, pp. 471-472, Zadar, 2010 (ISI).

A. Moldoveanu, F. Moldoveanu, V. Asavei, ***A. Morar****, A. Egner,* "From HTML to 3DMMO – A Roadmap Full of Challenges", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.620-627, Bucharest, 2011.

A. Egner, F. Moldoveanu, A. Moldoveanu, V. Asavei, ***A. Morar****,* "Automated Generation of TTCN-3 Type System Used for Testing of Healthcare Applications", Proceedings of the 18th International Conference on Control Systems and Computer Science, pp.794-799, Bucharest, 2011.

Research projects:

- EUREKA research project: **ReTeMes ("Reliability Testing of Medical Systems" – "Testarea fiabilitatii sistemelor medicale")**: research assistant (2009-2010)
- Leonardo da Vinci research project: **SMECluster ("Strategic Planning for Sustainable Clustering of Collaborative SMEs")**: research assistant (2009-2010)
- Leonardo da Vinci research project: **SMENet ("Establishment of Sustainable Collaborative SME Networks")**: research assistant (2009-2010)
- PNCDII – Partnerships research project: **SABIMAS ("Sistem informatic avansat, bazat pe imagistica medicala, pentru producerea implanturilor personalizate dedicata artroplastiei de sold - Advanced Information system, based on medical imaging, for personalized implants production in hip arthroplasty")**: research assistant (2009-2011)
- EuroStars Innovation research project: **RELIS ("Risk Detection in Laboratory Information Systems")**: research assistant (2010-2012)
- EuroStars Innovation research project: **EUGEN ("Enterprise Unified Guideline Engine")**: research assistant (2010-2012)
- EuroStars Innovation research project: **VISUAL-D ("Visualization of Patient Data for Easy Management of Care Processes")**: research assistant (2011-present)
- EUREKA research project: **MORIS F.D. ("Medical Operational Risks Identification Service and Fraud Detection" – Identificarea riscurilor medicale operationale si detectia fraudelor")**: research assistant (2011-present)
- Research project supported by the Austrian Science Fund: **KASI ("Knowledge Assisted Sparse Interaction for Peripheral CT-Angiography")**: research assistant during doctoral mobility (2012)

REFERENCES

[1] *W. E. Lorensen, H. E. Cline*, "Marching Cubes: a High Resolution 3D Surface Construction Algorithm", in SIGGRAPH '87 Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 163-169, New York, 1987.

[2] *P. Botez*, Ortopedie, Bit Publishing House (Iasi), 2001.

[3] *R. Kennon*, Hip and Knee Surgery: A Patient's Guide to Hip Replacement, Hip Resurfacing, Knee Replacement & Knee Arthroscopy, Middlebury, 2008.

[4] *R. C. Gonzales, R. E. Woods*, Digital Image Processing, Prentice-Hall, 2002, pp. 222-224.

[5] *J. Weickert*, Anisotropic Diffusion in Image Processing, B.G. Teubner Stuttgart, 1998.

[6] *S. Tabik, E.M. Garzon, I. Garcia, J.J. Fernandez*, "Implementation of anisotropic nonlinear diffusion for filtering 3D images in structural biology on SMP clusters", in Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of the International Conference ParCo, Vol. 33, pp. 727-734, 2005.

[7] *P. Perona, J. Malik*, "Scale-space and edge detection using anisotropic diffusion", in Proceedings of IEEE Computer Society Workshop on Computer Vision, pp. 16-22, 1987.

[8] *Canny J.F.*, "A Computational Approach to Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-8, Issue 6, pp. 679-698, 1986.

[9] *Y. Boykov, O. Veksler*, "Graph cuts in vision and graphics: theories and applications", in Math. Models of C. Vision: The Handbook, Springer, 2006.

[10] *M. Kass, A. Witkin, D. Terzopoulos*, "Snakes: active contour models", in International Journal of Computer Vision, pp. 321-331, 1988.

[11] *T. F. Chan, L. A. Vese*, "Active contours without edges", in IEEE Transactions on Image Processing, Issue 2, pp. 266-277, 2001.

[12] *H. Digabel, C. Lantuejoul*, "Iterative Algorithms", in Actes du Second Symposium Europeen d'Analyse Qantitative des Microstructures en Sciences des Materiaux, Biologie et Medicine, pp. 85-99, 1978.

[13] *S. Beucher, C. Lantuejoul*, "Use of watersheds in contour detection", in Proc. International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation, 1979.

[14] *J. B. T. M. Roerdink, A. Meijsetr*, "The watershed transform: definitions, algorithms and parallelization strategies", in Fundamenta Informaticae, Issue 41(1-2), pp. 187-228, 2000.

[15] *P. Porwik, A. Lisowska*, "The Haar-wavelet transform in digital image processing: its status and achievements", in Machine Graphics & Vision, vol. 13, no. 1/2, pp. 79-98, 2004.

[16] *A. Gavlasova, A. Prochazka, M. Mudrova*, "Wavelet based image segmentation", in Proc. of the 14th Annual Conference Techincal Computing, Prague, 2006.

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

[18] *M. Levoy*, Efficient Ray Tracing of Volume Data. ACM Transactions on Graphics, Vol. 9, Issue 3, pp. 245-261, 1990.

[19] *R. V. Pelt , A. Vilanova ,H. M. M. V. D. Wetering*, "GPU-based Particle Systems for Illustrative Volume Rendering", Volume Graphics, pp. 89-96, 2008.

[20] *F. Dong, G. J. Clapworthy, H. Lin, M. A. Krokos*,"Non-photorealistic Rendering of Medical Volume Data", IEEE Computer Graphics and Applications, Vol. 23, Issue 4, pp. 44-52, 2003.

[21] *X. Yuan, B. Chen*, "Illustrating Surfaces in Volume", in Proceedings of VisSym'04 Joint IEEE/EG Symposium on Visualization, 2004.

[22] *C. S. Co, B.Hamann, K.I. Joy*,"Isosplatting: A point-based alternative to isosurface visualization", in Proceedings of 11th Pacific Conference on Computer Graphics, pp. 325-334,2003.

[23] *Y. Luo, J. A. I. Guitian, E. Gobbetti, F. Marton*, "Context Preserving Focal Probes for Exploration of Volumetric Medical Datasets", in Proceedings of 3DPH, pp. 187-198, 2009.

[24] *G. Kindlmann, R. Whitaker, T. Tasdizen, T. Moeller*, "Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications", in VIS '03 Proceedings of the 14th IEEE Visualization, pp.513-520, 2003.

[25] *S. Bruckner, M. E. Gröller*, "Style Transfer Functions for Illustrative Volume Rendering", in Computer Graphics Forum, Vol. 26. Issue 3, pp. 715-724, 2007.

[26] *M. Hadwiger, C. Sigg, H. Scharsach, K. Buehler, M. Gross*, "Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. Computer Graphics Forum, Vol. 24, Issue 3, pp. 303–312, 2005.

[27] *M. Burns, J. Klawe, S. Rusnikiewicz, A. Finkelstein, D. DeCarlo*, "Line Drawings from Volume Data", in ACM Transactions on Graphics (Proc. SIGGRAPH), Vol. 24, Issue 3, pp. 512-518, 2005.

[28] Nvidia Corporation, NVIDIA CUDA C Programming Guide, Version 4.0, 2011.

[29] Nvidia Corporation, CUDA SDK Example, available at www.nvidia.com last visited in 18.05.2012.

[30] *J. Fung* , GPU Gems, Computer Vision on the GPU, pp. 649-664, 2005.

[31] *Y. Luo, R. Duraiswami*, "Canny Edge Detection on NVIDIA CUDA", in Proceedings of IEEE Computer Vision and Pattern Recognition Workshops, pp. 1-8, 2008.

[32] *G. J. Braak, C. Nugteren, B. Mesman, H. Corporaal*, "Fast Hough Transform on GPUs: Exploration of Algorithm Trade-offs", in Proceedings of Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS' 11), pp. 611-622, 2011.

[33] *M. Bojsen-Hansen*, "Active contours without edges on the GPU", in Project Paper for the Course in Parallel Computing for Medical Imaging and Simulation, 2010.

[34] *P. Harish , P. J. Narayanan*, "Accelerating large graph algorithms on the GPU using CUDA", in HiPC'07 Proceedings of the 14th international conference on High performance computing, pp. 197-208, Springer-Verlag Berlin, Heidelberg, 2007.

[35] *A. Kanitsar* , Curved Planar Reformation for Vessel Visualization, Phd Thesis, 2004.

[36] *T. Lee, R. Kashyap, C. Chu*, "Building Skeleton Models via 3-D Medial Surface/Axis Thinning Algorithms", Graphical Models and Image Processing, Vol. 56, Issue 6, pp. 462-478, 1994.

[37] *H. R. Nagel*, "GPU optimized Marching Cubes algorithm for handling very large, temporal datasets", CiteSeerX – Scientific Literature Digital Library and Search Engine, 2010.

[38] DICOM standard, available at http://medical.nema.org/standard.html - last visited in 31.08.2012

[39] SABIMAS project, available at http://graphics.cs.pub.ro/SABIMAS/ - last visited in 31.08.2012

[40] AngioVis project – Angiographic Visualization, available at www.angiovis.org – last visited in 31.08.2012

[41] *S. Chen, H. Jiang*, "Accelerating the Hough Transform with CUDA on Graphics Processing Units", Proceedings of 2011 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Pafos, 2011.

[42] *A. Kanitsar, D. Fleischmann, R. Wegenkittl, P. Felkel, E. Gröller*, "CPR – Curved Planar Reformation", Proceedings of IEEE Visualization 2002, pp. 37-44, October 2002.

[43] *A. Kanitsar*, Advanced Visualization Techniques for Vessel Investigation, Master thesis, Vienna University of Technology, Institute of Computer graphics and Algorithms, March 2001.

[44] *D. Mumford, J. Shah*, "Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems", in Communications on Pure and Applied Mathematics, Vol. 42, Issue 5, pp. 577-685, July 1989.

[45] *A. Haar*, "Zur Theorie der orthogonalen Funktionsysteme". Mathematische Annalen, Vol. 69, Issue 3, pp. 331-371, 1910.

[46] *P. S. Moharir*, Pattern Recognition Transforms, New York: John Wiley, 1992.

[47] *P. Shirley, A. Tuchman*, "A Polygonal Approximation to Direct Scalar Volume Rendering", Computer Graphics, Vol. 24, Issue 5, pp. 63-70, 1990.

[48] *J. E. Roos, D. Fleischmann, A. Köchl, M. Straka, A. Napoli, A. Kanitsar, M. Sramek, E. Gröller*, "Multipath Curved Planar Reformation of the Peripheral Arterial Tree in CT Angiography", Radiology, Vol. 244, Issue 1, pp. 281-290, 2007.