Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

An Interactive Framework for Globally Optimal Image Segmentation with Local Constraints

Markus Unger

Graz, Austria, January 2008

Supervisor

Univ. Prof. DI Dr. Horst Bischof

Instructor

DI Thomas Pock

Abstract

Image segmentation is a challenging task in computer vision. We present a general purpose image segmentation framework, and focus on its application to medical imaging. Features like gray values or edges are commonly used as input for segmentation algorithms. The geodesic active contour model gained popularity as a flexible variational image segmentation model based solely on edge information. Unfortunately the geodesic active contour model exhibits local minima, making segmentation results strongly dependent on its initialisation. We propose a globally optimal segmentation model, that unifies the usage of gray value information with the geodesic active contour model. A flexible, interactive segmentation framework is presented, that allows incorporation of local constraints. Fast numerical schemes are used to minimise the proposed energy which is based on a weighted Total Variation energy functional. Different segmentation approaches using the proposed energy functional are discussed. The relation to the image denoising task is analysed, and we present a fast implementation of the image denoising model of Rudin, Osher and Fatemi. With our GPU-based implementation real-time performance is achieved for both 2D and 3D segmentation problems. We show experimental results on various real world images and different medical datasets.

Keywords. Segmentation, Total Variation, Globally Optimal GAC, GPU, Denoising, Local Constraints

Contents

Chapter 1

Introduction

Contents

1.1 Problem Statement

This master's thesis addresses the problem of image segmentation. Image segmentation is a fundamental problem in computer vision, that tries to partition an image into disjoint regions. Many different approaches exist. Various kinds of information, like gray values, edges or any other local features, can be used as input for the segmentation. A common drawback of a wide range of segmentation algorithms is that they lack generality. Although they perform very well on a specific task, they can only be used on a very small class of segmentation problems. The goal of this master's thesis is to develop a versatile segmentation framework, that can be used for a wide range of input images. The proposed segmentation algorithm relies both on gray value and on edge information. Additionally user information is accounted for. Although the proposed algorithm is fully designed and applicable as a general purpose segmentation tool, we will focus on its application to medical data. Medical imaging is a challenging task, and a lot of work was done on medical image segmentation.

Snakes [50] and geodesic active contours [21] showed promising results on segmentation tasks. Unfortunately most of the existing algorithms that solve this models, get stuck in local minima. The segmentation algorithm presented in this master's thesis, is able to compute globally optimal geodesic active contours as a special case. Another common problem is the incorporation of high level knowledge in the segmentation process. We will address this problem by allowing different local constraints.

The proposed segmentation model has a profound mathematical framework. It is based on the Total Variation and energy functionals. The minimisation of the energies forms an important part of this master's thesis. Although fast numerical methods exist, they are still computationally expensive. We will make use of the parallelisation potential of the algorithms, and present a fast implementation on the graphics device.

The proposed segmentation model is closely related to the image denoising model by Rudin, Osher and Fatemi [76]. Image denoising is another basic task in computer vision, and aims to reconstruct the original image that is corrupted by noise. We will discuss the relation to the segmentation task, and also show results using a fast implementation.

1.2 Outline

In Section 1.3 we first discuss common medical image modalities. As we focus mainly on medical images one could gain a better insight to artifacts by understanding the underlying acquisition mode. In Section 1.4 the general segmentation problem is discussed, and common approaches to image segmentation are reviewed.

Section 2 focuses on recent work that is closely related to the proposed segmentation model. In Section 2.1 we review deformable models, namely the snake or active contour model [50], geodesic active contours [21]. Section 2.2 focuses on different solutions for the models presented in Section 2.1. We discuss pros and cons, and review the methods used to solve the segmentation models. In Section 2.3 the Total Variation approach is discussed. We review different denoising models, and show the relation to image segmentation. Previous approaches using the Total Variation are discussed in Section 2.3.3. We will discuss the Mumford Shah functional [63] and the g-weighted Total Variation [16]. Different solutions to the denoising problems are derived in Section 2.4. Various existing approaches are discussed. In this master's thesis, the primal approach in Section 2.4.2 and the dual methods presented in Section 2.4.3 were implemented for evaluation purposes.

In Section 3 the proposed segmentation method is presented. Section 3.1 focuses on the proposed segmentation algorithm. We introduce the segmentation energy, and then show a way to minimise the energy by applying the same methods as for the denoising algorithms. Furthermore we show the relation of our algorithm to the maximal flow algorithm of Appleton and Talbot [4] in Section 3.1.3. Numerical methods, as used for implementation, are derived in Section 3.2.

In Section 4 the implementation on graphics hardware is discussed. We address the benefits of state-of-the-art graphics hardware in Section 4.1. The used programming language CUDA is coupled with the newest generation of GPUs, and will be discussed in Section 4.2. The parallelisation potential of the algorithm makes implementation on the GPU very fast. Performance issues are addressed in Section 4.3. We also discuss the graphical user interface used for interaction with the algorithm in Section 4.3.2.

Section 5 presents results of the implemented algorithms. Speed evaluations for the image denoising tasks can be found in Section 5.2. An evaluation of the proposed segmentation algorithm was done in Section 5.3. Segmentation results in two and three dimensions are presented.

In Section 6.1 we give a short conclusion, and future work is discussed in Section 6.2.

1.3 Image Modalities

The segmentation approach presented here is intended to be very flexible allowing usage in several segmentation problems. We primarily focus on medical images, especially 3D volume data as this is an ongoing and challenging task with a lot of promising applications. Therefore common medical imaging techniques and the problems they bring along are discussed in the following. More detailed introductions to medical imaging can be found in [37, 84].

1.3.1 Common Imaging Artifacts

The most common imaging artifact is noise, a random value $v(x, y)$ that is added to the pixel intensity. Noise leads to additional intensity variations within homogeneous regions and disturbs edge detection by influencing the image gradients. Gaussian filtering is one of the most simple approaches to reduce noise. When dealing with salt and pepper noise median filtering is more effective. Usually the image is also degraded by a known linear operator k caused by the image acquisition system. Therefore the observed image $f(x, y)$ can be written as

$$f(x, y) = (u * k)(x, y) + v(x, y) \ . \tag{1.1}$$

There are also a lot of nonlinear imaging artifacts one has to deal with. An effect that occurs in all image modalities is the partial volume effect. As illustrated in Figure 1.1 it

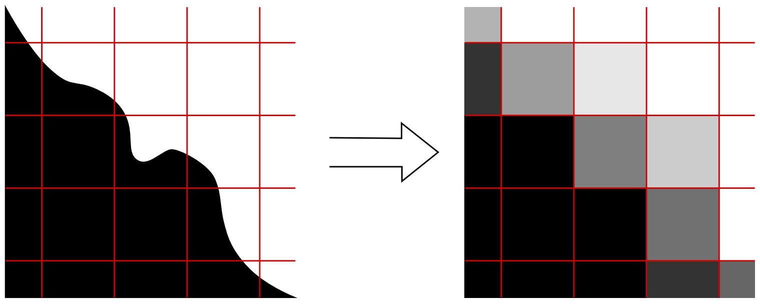
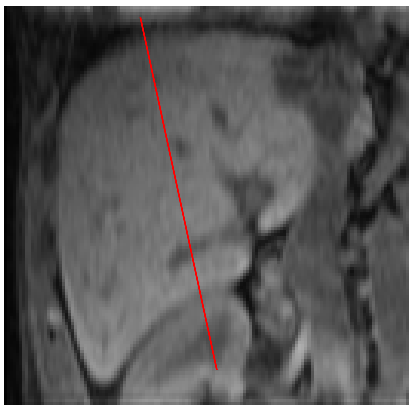


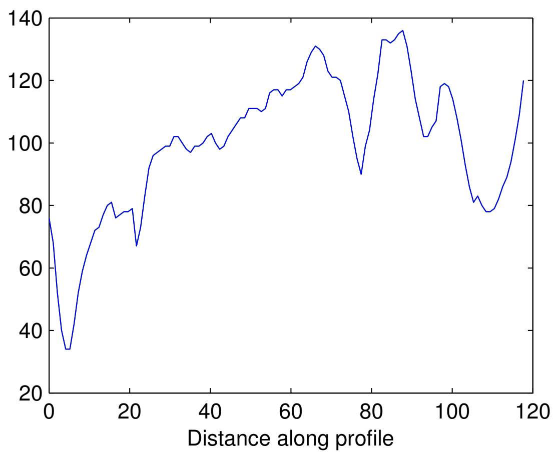
Figure 1.1: Partial Volume Effect: Voxels with different kind of tissue are averaged.

occurs when different kinds of tissue lie in a single voxel. This results in an averaged value that usually does not correspond to any of the tissue characteristics. The partial volume effects locally degrades edge quality that can be important for a good segmentation. The partial volume effect can be approximated by the linear operator k.

Another common problem emerges from intensity inhomogeneities as illustrated in Figure 1.2. Intensity gradients are often not avoidable as they are caused by different thickness of tissue and by changes during the acquisition process. Inhomogeneities in intensities make basic threshold segmentation quite useless.



(a)



(b)

Figure 1.2: (a) MR image of the liver. (b) Gray-scale intensity profile along the red line.

1.3.2 X-Ray Image Generation

X-ray image generation relies on the attenuation effect of x-rays as they travel through the body. In medical applications the wavelength of x-rays lies in the range from 8 to 50 pm (25 - 150 keV). X-rays are an ionising radiation and too high doses can be harmful to the human body. The pixel intensity is proportional to the attenuation coefficient of the tissue, where each pixel represents the integral of the attenuation coefficients along the beam. Thus the gray-level depends not only on the type of tissue but also on thickness, leading to strong intensity inhomogeneities. As x-ray images have good spatial resolution and low noise, they are an eligible input for image segmentation, especially when one is interested in bones that provide strong contrast.

1.3.3 Computer Tomography

Computer Tomography (CT) is also based on x-rays but provides cross-sectional images. This is achieved by rotating source and detector around the patient. From these integral images the true voxel intensities are calculated. CT imaging provides 3D images with a good spatial resolution of typically 0.5 mm and a strong bone contrast. Scanners have already gotten very fast, and it is even possible to capture a beating heart. Unfortunately the x-ray dose is very high, why utilisation of CT should be carefully considered.

1.3.4 Magnetic Resonance Imaging

In magnetic resonance imaging (MRI) one makes use of nuclear magnetic moments of typically hydrogen atomic nuclei. The otherwise chaotic spins of the nuclei are aligned by a strong, constant magnetic field, leading to a magnetisation vector. Through a high frequency pulse the spins are synchronised and flipped by 90°, making it possible to measure the magnetisation vector by an induction coil. One can now measure spin-lattice relaxation time T_1 that is based on flipping back of the spins by transmitting energy to the neighbourhood, which depends on hydrogen concentration and the chemical composition of the neighbourhood. Furthermore one can determine the spin-spin-relaxation time T_2 based on dephasation of the spins caused by alternation of the magnetic field of the neighbourhood spins. An extensive overview on MRI can be found in [45]. MRI allows to improve differentiation between various kinds of tissue by adjusting parameters for a specific task. It delivers good image quality with reasonable spatial resolution of about 1 mm. Furthermore it has a low signal-to-noise ratio and is assumed to be no danger to health. Drawbacks are currently only the high price and long examination time.

1.3.5 Ultrasound Image Generation

Ultrasound imaging relies on longitudinal mechanical waves in the range from 1 to 20 MHz. The pulse-echo method sends out a short ultrasound wave that penetrates the body. Due to the fact that different kinds of tissue have a different characteristic impedance, reflections occur at tissue boundaries. The strength of these echos directly correspond to gray-scale intensity, and the travel time yields the depth where the reflection occurred. In medical imaging the B-mode, that produces a 2D slice of the body, is very common, but also Doppler imaging and recently 3D image acquisition modes get important. The main advantages of ultrasound imaging are its simplicity, low cost and its harmlessness regarding health. Unfortunately image quality is not that good as there are many artifacts caused by multiple echos, distortions due to different sonic speeds, refraction, shadowing behind bone, side lobes of the beam and speckle. Therefore ultrasound images are very noisy, have low contrast and contain weak and incomplete border information, making segmentation a very difficult task. In [67], a comprehensive survey on segmentation approaches in ultrasound imaging is given.

1.3.6 PET - SPECT

Positron emission tomography (PET) and single photon emission tomography (SPECT) are functional imaging modalities that allow to capture 3D volumes of metabolic activity. In PET scans positron emitting radioactive isotopes are introduced into the body. The positrons are detected and an image is generated that has a good spatial resolution of about 4-5 mm. Unfortunately the production of isotopes and marking of tracers is very time critical and expensive. SPECT on the other hand use the decay of radioisotopes that emit photons that can be detected. This solution is much cheaper and more flexible, but has a spatial resolution of about 5-15 mm. Both image modalities usually deliver very noisy images.

1.4 Segmentation

The problem of image segmentation is one of the most fundamental problems in computer vision. Segmented images are used as input for various applications such as classification, recognition and measurement, or specifically in the case of medical image processing for the study of anatomical structures, diagnosis and the planning of surgeries or other forms of treatment. The main objective of image segmentation consists in splitting up an image into several disjoint regions. Given an image I the complete segmentation problem is to determine a set of K disjoint regions $R_1, \ldots, R_K \subset I$ such that

$$I = \bigcup_{k=1}^{K} R_i \qquad R_i \cap R_j = 0 \quad for \quad i \neq j \; . \tag{1.2}$$

These regions shall correspond to real world objects or parts of them. It is obvious that the problem of image segmentation is highly ambiguous. In medical applications the region of interest is usually a distinct anatomical structure. Using prior knowledge such as shape information, it is possible to specialise an algorithm to find a certain object. On the other hand one can design an algorithm more flexible, allowing segmentation of arbitrary objects. Consequently, such an algorithm will highly depend on its parametrisation and requires some form of user interaction. It is the aim of any segmentation algorithm to keep interaction as little as possible.

There are several issues like noise, acquisition artifacts, poor contrast, weak boundaries, inhomogeneities and distracting textures that make image segmentation a very difficult task. Especially in medical imaging problems like partial volume effects and intensity inhomogeneities are very common as already discussed in Section 1.3.1. In the following, we will review some common image segmentation techniques and will show their limitations. For a more detailed description of basic segmentation approaches see for example [83], or in a more medical context [53, 71, 82, 84].

1.4.1 Thresholding

The simplest approach towards image segmentation is gray-level thresholding. Thresholding is also a kind of region based approach. In case the desired objects gray levels differ strongly from those of the background, thresholding delivers quite good results. One can use one or more threshold values that delimit a class from the other. Generally we can write:

$$g(x,y) = \begin{cases} 1 & for\ I(x,y) \in D_1 \\ 2 & for\ I(x,y) \in D_2 \\ \ldots & \\ n & for\ I(x,y) \in D_n \\ 0 & else \end{cases} \tag{1.3}$$

with a gray-level set D_i for each class, and $D_i \cap D_j = 0$ for $i \neq j$. Equation 1.3 applies to all global thresholding algorithms, but there are also local adaptive thresholding shemes where D_i is depending on spatial location.

One can choose the threshold manually, or use an automatic approach. Automatic methods for threshold detection commonly rest upon histograms. Basically one can decide between shape-based histogram techniques, where features of the histogram are used to determine the thresholds, or optimal thresholding. An exhaustive survey on thresholding techniques and their evaluation is given in [78]. They classify the thresholding algorithms into the following six categories: (1) Histogram shape-based methods, which use features of the histogram like the position of valleys or maximum curvature. (2) Clustering based thresholding methods, where one tries to find two clusters e.g. by fitting mixtures of Gaussians. (3) Entropy-based thresholding methods try to make use of the entropy of foreground and background, or the cross-entropy between the original and the segmented image. (4) Thresholding based on attribute similarity, that are based upon a similarity measure between the original image and the binarized version, like edge matching, moments, texture, stability, connectivity or shape compactness. (5) Spatial Thresholding Methods also incorporate dependency of pixels in a local neighbourhood and make use of probability distributions and correlation between pixels. (6) Local adaptive methods define a unique threshold for every pixel depending on the local characteristics of the image.

Most of these algorithms can be implemented quite fast and are suitable for real-time applications. When using x-ray or MR imaging it will be easy to separate bones from fat. But especially when using global approaches it is illusionary to separate all kinds of tissue, as the corresponding gray-level sets are overlapping in wide areas. In Figure 1.3 an example is given, where the segmentation of the bones in a x-ray image of the hand failed. Thresholding approaches are also very sensitive to noise. Furthermore they lack the possibility of regularisation. Therefore simple thresholding algorithms are not suited for medical applications, but are often used as input for further processing.

1.4.2 Region Based

Region based approaches extract regions that satisfy a homogeneity criteria like gray-level, colour, texture, shape, spatial location and many more. Therefore all regions have to satisfy a binary homogeneity criteria $H(R_i)$. We can thus extend Equation 1.2 by the following conditions:

$$H(R_i) = TRUE \quad for \quad i = 1, \dots, K \ , \tag{1.4}$$

$$H(R_i \cup R_j) = FALSE \quad for \quad i \neq j \ . \tag{1.5}$$

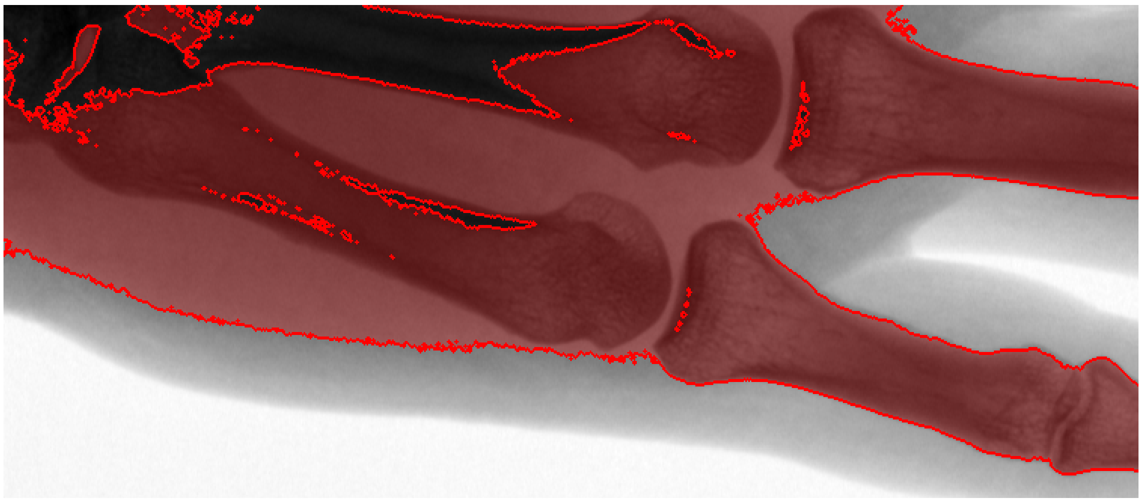


Figure 1.3: Failed threshold segmentation of the *Bone* image.

A very simple approach is region growing. One starts with one or more midpoints, and applies an iterative algorithm. First all neighbours around the border are evaluated whether they satisfy the homogeneity criteria and if they do, they are added to the region. Iterations continue until no changes occur. Region growing often results in too many small regions, and therefore most of the time post-processing is necessary. Furthermore leakage is a great problem. Region merging connects two regions so that they satisfy a common homogeneity criteria. On can perform region merging based on the output of a region growing algorithm or start with an arbitrary initialisation of regions. Region splitting is the opposite and usually begins with the whole image as one region that will most probably not satisfy conditions 1.4 and 1.5. Therefore the existing regions are split into smaller ones, until the homogeneity criteria is satisfied. Splitting and merging are often combined as it was already described in [46]. The split-and-merge approach starts with the whole image as one region. If they do not satisfy the homogeneity criteria they are split into four child-regions. If different child-regions satisfy the same homogeneity criteria they are merged. When no regions can be split or merged this way, adherent regions that satisfy the same homogeneity criteria are merged.

Of course also edge information can be incorporated into the region based approaches. Another region based approach is the concept of watershed segmentation. Here the image data is interpreted as a topographic surface, that is flooded by water. An efficient algorithm was first proposed by Vincent and Soille in [88]. Starting with the lowest gray values as initial seed-points for regions the catchment basins are flooded. If two catchment basins are to merge an artificial wall, the watershed, is build up to the highest altitude. These watersheds represent the segmentation of the image. Usually the gradient image is used as input for the algorithm, but sometimes it is useful to apply a distance transformation first, allowing to separate touching objects. Automatic watershed segmentation usually results in strong over-segmentation, that can be improved by manually defining starting regions for the algorithm. Watershed segmentation is well suited for segmentation of objects with homogeneous gray-values. In [82], an example of prostate segmentation in MR images and segmentation of the left ventricle in tagged MR images using the watershed algorithm is given.

Another segmentation approach makes use of the Maximally Stable Extremal Region (MSER) detector. This detector performs very well on various imaging tasks. In [36], a segmentation algorithm using a slightly modified MSER detector is presented. The algorithm needs a seed point and performs quite well on colour images.

1.4.3 Edge Based

Edge based image segmentation methods utilise edge information to split an image into regions. To obtain an edge image one can make use of the first-order derivative (e.g. Sobel, Prewitt operator). A very common but more complex algorithm that yields very good results is the Canny edge detection [19]. Also very common is the Marr-Hildreth or Laplacian-of-Gaussian (LoG) algorithm that utilise the second-order derivative. The resulting edge images often contain gaps or a lot of small and weak edges. One can use edge relaxation techniques to improve the detection by evaluating the edges in context of their neighbourhood. As we have no segmentation yet, edge images are always used as input for further processing.

One method to obtain a segmentation is border tracing. One tries to follow the border by some predefined rules. Usually these simple methods have several drawbacks as they get stuck in dead ends and are not able to find global optimal solutions. One can overcome these problems by using graph searching. Graph searching is a method used in many disciplines and tries to find an optimal path between a start and an end point. The edge image is used as a cost function where strong detections result in low costs. Thus it is possible to compute optimal paths in a graph that define the segmentation of the image. Dynamic programming follows a similar idea. It is based on the principle of optimality [8] stating that a short path of an optimal way through a graph has to be optimal too. Implementation is achieved by computing a cumulative cost matrix based upon a static cost matrix, the edge image. In this cumulative cost matrix the optimal border is traced back from the end point to the start. For these methods some prior knowledge is necessary as one needs a start and an end point. These can either be selected manually or by automatic methods. When shape and size of the objects are known, the Hough transformation [47] can be used for segmentation. The key idea is to transform an image to a parameter space where all points on an object are mapped into the same point. For objects where no parametric representation is available, the generalised Hough transformation can be used. In Section 1.4.6 more segmentation methods that make use of a priori knowledge and object models will be presented.

1.4.4 Classification

Image segmentation methods using classification partition a feature space that is derived from the original image. There are many possible features like colour, texture, gradients, or for example in MRI imaging T1 and T2 weighting or other image modalities. Also spatial information can be incorporated into classification algorithms. One discriminates between supervised and unsupervised techniques. Supervised classification algorithms require a set of samples for each class, usually forcing the user to provide hand-labelled data. On the other hand unsupervised classification algorithms perform cluster analysis to find the underlying structure of the image.

Examples for supervised classification techniques are minimum distance, maximum likelihood (ML), Bayes and k-nearest-neighbour classifiers. As they are hardly used for image segmentation, we will not discuss them here. Unsupervised techniques are more common in image segmentation, and a general overview on different classification algorithms can be found in [9, 49, 91]. The k-means algorithm tries to minimise the in-class scatter. This is usually done by an iterative algorithm: Based on an initial selection of class centres, using any distance measure the closest data points are assigned to a cluster. The centre of the cluster is recalculated as long as changes happen. One great disadvantage is the fixed number of classes. The ISODATA clustering algorithm improves this by allowing splitting and merging of classes. With fuzzy c-means clustering soft segmentations where one data point is assigned to different classes are allowed. The expectation-maximisation (EM) algorithm performs clustering based on the assumption of an underlying Gaussian mixture model for the data. All this algorithms are very sensitive to initialisation.

Another clustering algorithm that proved very well in image segmentation is the mean-shift segmentation [34]. Here a kernel is moved through the data according to its current mean. All data points once in the kernel will be assigned to a common class. Kernels that end up in the same point are merged. Mean-shift segmentation is very flexible, but requires careful selection of the kernel. It has already been successfully used for medical applications [59].

Also clustering approaches exist that make use of artificial neural networks. All these clustering algorithm allow for multiple spatial not connected regions in the segmentation. Unfortunately parametrisation is not trivial in most of the cases.

1.4.5 Atlas-guided

In atlas-guided approaches segmentation is treated as a registration problem. There exits a huge number of registration algorithms [57]. The key idea is that there already exists a correct segmentation for the atlas or reference image. By finding the mapping of the image to the atlas one also gets the mapping of the segmentation to the image. Atlas-guided approaches are widely used in medical imaging [74]. Recent work was done for example with caudate segmentation in [41]. While the algorithms work well on structures with little variability, they perform very poor when great changes in structure or topology occur. As non-rigid registration implementations are computationally very expensive, atlas-guided segmentations in 3D are usually slow.

1.4.6 Deformable Models

Deformable models are curves or surfaces that are influenced by internal and external forces. The internal forces maintain the smoothness of the model during the deformation process. The external forces move the model according to features defined by the data, e.g. forcing the model boundaries towards the edges.

One of the earliest approaches is the snake or active contour model by Kass, Witkin and Terzopoulos in [50]. Snakes are a parametric contour model that is represented using polygons or splines. Later the geodesic active contour (GAC) model was developed by Caselles et al. [21]. This model is intrinsic, as it does not depend on the curve parametrisation. These models will be reviewed in depth in Section 2.1, and possible approaches to solve this models will be discussed in Section 2.2. We will also show that the segmentation approach taken in this paper is closely related to the GAC model.

There exist a lot of extensions to the basic deformable models. Most of them deal with the incorporation of prior information such as shape. Shape information can be very useful when the variations within the population are rather small. But they narrow the segmentation approach towards a single class of objects. We will not discuss them here, as we aim to build a very flexible segmentation tool. But there exists a lot of literature on this topic. For an overview see e.g. [82] or [15]. In [62], a survey on deformable models in medical image analysis is given.

Slightly different to the above approaches, the active shape models were proposed by Cootes et al. [35]. They use prior models that are based on a set of points defined at special features of the image.

Chapter 2

Related Work

Contents

2.1 Deformable Models

Deformable models for image segmentation had a great influence on segmentation techniques. Contours are deformed according to internal and external forces. A possible way deformable models can be described, are energy functionals. We will first review the snake / active contour model by Kass et al. [50], and then the extension to geodesic active contours that were developed by Caselles et al. in [21].

2.1.1 Active Contours

Snakes are one of the earliest active contour models and were introduced by Kass, Witkin and Terzopoulos in [50]. The contour is explicitly modelled as a spline, and is deformed by various forces to find a segmentation of the image. Mathematically the parametric curve $C(s) = (x(s), y(s)) \in \Omega, s \in [0, 1]$ is moved through the image domain Ω to minimise the following energy functional proposed by Kass et al.:

$$E_{snake} = \int_0^1 E_{int}(C(s)) + E_{ext}(C(s)) + E_{con}(C(s))ds \; . \tag{2.1}$$

Here E_{int} represents the internal energy of the snake maintaining smoothness and tension. E_{ext} is the external energy and is derived directly from the image. To allow user interaction or higher level information the term E_{con} can incorporate constraints into the energy functional.

The internal energy is, corresponding to [50], defined as

$$E_{int}(C(s)) = \alpha(s)\left|\frac{\partial C(s)}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2 C(s)}{\partial s^2}\right|^2 . \tag{2.2}$$

Here the first-order derivative makes the curve act like an elastic string. The second-order derivative makes the curve behave like a thin plate. Thus $\alpha(s)$ controls the 'tension' and $\beta(s)$ the 'rigidity'. Usually they are chosen as constants. Setting $\beta = 0$ allows for corners in the curve.

The external energy E_{ext} shall depend on the image data $I(x, y)$, and is therefore defined as

$$E_{ext} = P(I(C)) . \tag{2.3}$$

To attract the curve towards edges the potential P is for example chosen as

$$P(x, y) = -\lambda \left|\nabla \left[G_\sigma * I(x, y)\right]\right| , \tag{2.4}$$

with ∇ the gradient operator, and $G_\sigma * I(x, y)$ is the image convolved with a Gaussian kernel, representing a smoothed version of the image. λ is a scale parameter that determines the influence of the image data. The curve will always be attracted by low energy, and therefore any other potential function P that vanishes in flat regions is possible.

The problem of minimising the energy can be solved by the principle of the calculus of variation. The curve $C(s)$ that minimises E_{snake} must satisfy the following Euler-Lagrange equation:

$$-\frac{\partial}{\partial s}\left(\alpha\frac{\partial C}{\partial s}\right) + \frac{\partial^2}{\partial s^2}\left(\beta\frac{\partial^2 C}{\partial s^2}\right) + \nabla P(C) = 0 . \tag{2.5}$$

In [50], a solution to solve Equation 2.5 is given using a numerical finite differences method. The solutions of this PDEs will be discussed in Section 2.2.

The snake model by Kass et al. requires user interaction for selecting a model that lies close to the true segmentation. The user can modify points of the curve and forces in real-time, as implementations are very fast. Snakes can also be extended to surfaces in 3D. Unfortunately one can only obtain a local minimum as the energy E_{snake} is non-convex. The functional also depends on the parametrisation of the curve C and is thus not intrinsic. The greatest problem is that the model cannot change topology automatically. It is very time consuming to detect self-intersections of the contour, especially in 3D.

In [31], Cohen extended the snake model by incorporating a pressure force that makes the model behave like a balloon. This way the model will not get stuck at weak edges, and instead find only strong ones. In the beginning, segmentation of 3D volumes was done for each 2D slice separately. This sequence of segmentations was then connected to a single 3D surface [32]. Later true 3D models were developed as e.g. in [33], allowing much faster and robust methods. See [62] for a survey on this topic.

2.1.2 The Geodesic Active Contour Model

Geodesic active contours (GAC) in 2D and minimal surfaces in 3D were introduced by Caselles et al. in [21, 22] and simultaneously in [51, 52]. The GAC model is based on the active contour/snake model detailed in Section 2.1.1. The second-order smoothing term in Equation 2.2 is removed by setting $\beta = 0$. This term is redundant as the model will still decrease the curvature [21]. Thus the model of Equation 2.1 (not taking into account any user interaction) simplifies to:

$$E(C) = \alpha \int_0^1 \left| \frac{\partial C(s)}{\partial s} \right|^2 ds - \lambda \int_0^1 \left| \nabla I(C(s)) \right| ds \; . \tag{2.6}$$

The edge detector can be generalised by replacing $-\left|\nabla I\right|$ by $g\left(\left|\nabla I\right|\right)^2$ if g is a strictly decreasing function such that $g(x) \to 0$ as $x \to \infty$ e.g.:

$$g(I) = \frac{1}{1 + \delta \left|\nabla \left(G_\sigma * I\right)\right|^2} \; . \tag{2.7}$$

Where δ represents an arbitrary positive constant. The goal is to minimise Equation 2.6. But the formulation is still not intrinsic. In [21], Caselles et al. showed using concepts of Hamiltonian theory that the minimisation of the following formulation is an equivalent problem:

$$E_{GAC}(C) = \int_0^{|C|_{\mathcal{E}}} g\left(\left|\nabla I(C(s))\right|\right) dl \; . \tag{2.8}$$

$|C|_{\mathcal{E}}$ is the Euclidean length of the the curve C defined by $|C|_{\mathcal{E}} = \oint \left|\frac{\partial C(s)}{\partial}\right| ds = \oint dl$ and dl is the Euclidean element of length. Equation 2.8 sums up the Euclidean element of length dl weighted by a term that directly depends on the boundary information of the data. We thus have an intrinsic formulation of the problem that depends on the geometrical information of the data. Caselles et al. showed that this is equal to finding a geodesic curve in a Riemannian space. In a given Riemannian space the length of a contour is given by

$$|C|_{\mathcal{R}} = \int_0^{|C|_{\mathcal{E}}} \sqrt{\mathcal{T}^T D(C(s))\mathcal{T}}\ . \tag{2.9}$$

Where $\mathcal{T}$ is the unit tangent vector to the contour, and D specifies a local Riemannian metric at a given pixel. $E_{GAC}(C)$ and $|C|_{\mathcal{R}}$ are equal in the case of an Riemannian metric $D = \mathbf{diag}(g(|\nabla I|))$. In [5, 6], the equivalence of Equation 2.6 and 2.8 is analysed in depth.

To find a solution for minimising E_{GAC} the gradient decent method can be applied to the Euler-Lagrange equation:

$$\frac{\partial C(t)}{\partial t} = g(I)\kappa\mathcal{N} - (\nabla g \cdot \mathcal{N})\mathcal{N}\ . \tag{2.10}$$

Here κ is the curvature and $\mathcal{N}$ is the unit normal to C. Equation 2.10 provides the fastest way to decrease the energy E_{GAC}. To find a solution for this PDE, Caselles et al. used level set methods that will be described in Section 2.2.2.

The GAC model can deliver very good results, relies on a profound mathematical framework and has strong theoretical properties. Unfortunately in practise the segmentation result highly depends on the initialisation. Minimisation of the non-convex energy functional E_{GAC} usually get stuck in local minima. Therefore a global optimiser is desirable. But note that the trivial solution $C = 0$ is always a global optimiser of the GAC model. To account for this, high level information has to be incorporated, e.g. in form of constraints. Later, we will see that our approach is able to calculate globally optimal geodesic active contours with local constraints.

2.2 Finding a Solution

In the following we present recent work that aims to find solutions to the segmentation models described in the previous section. We will first discuss a basic explicit scheme to solve the snake model. For the GAC model the level set approach, graph based methods and the Continuous Maximal Flow algorithm by Appleton and Talbot are discussed.

2.2.1 Explicit Solution

Kass et al. already provided a solution to the snake model. In [50], he proposed a finite difference method as a solution. To solve Equation 2.5, the contour is made dependent on time $C(s,t)$. The right hand side of Equation 2.5 is then set to the partial derivative of C:

$$-\frac{\partial}{\partial s}\left(\alpha\frac{\partial C}{\partial s}\right)+\frac{\partial^2}{\partial s^2}\left(\beta\frac{\partial^2 C}{\partial s^2}\right)+\nabla P(C)=\gamma\frac{\partial C}{\partial t}\ , \tag{2.11}$$

where γ is a coefficient to make the units of both sides consistent. Equation 2.11 can be seen as a gradient decent algorithm. When the right term vanishes the evolution process stops, and we get a solution that represents a local minimum of Equation 2.1.

Kass et al. derived the following iterative update scheme

$$\boldsymbol{C}^n=(\boldsymbol{I}-\tau\boldsymbol{A})^{-1}\left[\boldsymbol{C}^{n-1}+\tau E_{ext}(\boldsymbol{C}^{n-1}\right]\ , \tag{2.12}$$

where $\boldsymbol{A}$ is a pentadiagonal banded matrix of the size $m\times m$ with m the number of sampling points, and $\tau=-\frac{\Delta t}{\gamma}$. Using LU decomposition it is easy to compute the inverse of the matrix $\boldsymbol{I}-\tau\boldsymbol{A}$ making the implementation reasonably fast, especially when the sampling intervals are big.

This method also works in 3D [86] but uses much more computational power. A lot of other numerical implementations exist. Some of them use finite element methods [61], dynamic programming [3] or the greedy algorithm [90].

2.2.2 Level Set Method

Level set methods were first introduced by Osher and Sethian in [70]. Since then they were widely used, and there exists a lot of literature [58, 77]. Level set methods implicitly represent a curve as a level set of a higher dimensional function. This function is usually defined on the same domain as the image, and is referred to as the level set function. A level set is defined as the set of points with the same function value. Instead of directly evolving the curve through time, the level set function is updated. This approach has the favourable property that the topology of the curve can change during the evolution process. These principles are shown in Figure 2.1. We will later show how to solve the curve evolution problem using the level set formulation.

Before concentrating on level sets, we have to make some general remarks on the evolution of curves. Notations are all done for the two dimensional case, but the curve C can be easily replaced by a surface of hyper-surface Γ. A curve C with its initial contour $C(t=0)=C_0$ has the following general evolution PDE:

$$\frac{\partial C}{\partial t}=V_{\|}\mathcal{T}+V_{\perp}\mathcal{N}\ . \tag{2.13}$$

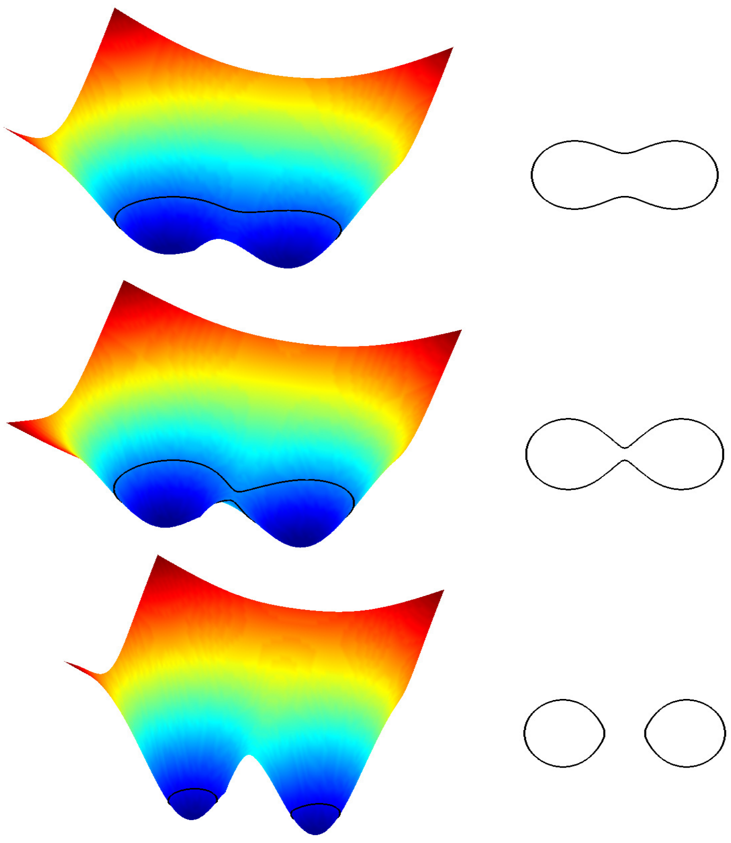


Figure 2.1: On the left hand side we see the evolution of the level set function Φ, and on the right hand side the evolution of the contour. One can see that the contour can change topology while the level set function remains still valid.

Here $\mathcal{T}$ is the tangential to C and $V_{||}$ the tangential velocity. $\mathcal{N}$ is the unit normal and $V_{\perp}$ the normal velocity. The deformation of the curve does not depend on the tangential velocity $V_{||}$ and is therefore dispensable. The normal velocity $V_{\perp}$ depends on the curvature, and we can write $V(\kappa)$ instead, leading to the following simplified PDE:

$$\frac{\partial C}{\partial t} = V(\kappa)\mathcal{N} \, . \tag{2.14}$$

As $V(\kappa)$ determines the speed of the evolution process it is called the speed function. The unit normal $\mathcal{N}$ and the curvature κ are given as

$$\begin{aligned} \mathcal{N} &= -\frac{\nabla\Phi}{|\nabla\Phi|} , \\ \kappa &= \nabla \cdot \mathcal{N} = \nabla \cdot \left(-\frac{\nabla\Phi}{|\nabla\Phi|} \right) . \end{aligned} \tag{2.15}$$

A very common speed function $V(\kappa) = \alpha\kappa$ is used in curvature deformation leading to the geometric heat equation $\frac{\partial C}{\partial t} = \alpha\kappa\mathcal{N}$ with a positive constant value α. Here the curve will be smoothed and shrink to a circular point. Another application is the constant deformation given by $\frac{\partial C}{\partial t} = v\mathcal{N}$ that has the effect of a pressure force. The coefficient v determines the direction and speed of the deformation. For geometric deformable models the speed function has to be coupled to the image data.

We now assume a level set function $\Phi(x, y, t)$ with its zero level set defining the contour $C(s, t)$:

$$\Phi(C(s,t), t) = 0 . \tag{2.16}$$

We define the level set function such that the values inside the zero level set are negative, and positive outside. The derivation of Equation 2.16 is given as

$$\frac{\partial\Phi}{\partial t} + \nabla\Phi \cdot \frac{\partial C}{\partial t} = 0 . \tag{2.17}$$

As for the dot product, only normal velocities are relevant for the evolution process. The evolution process of the zero level set can simply be written as

$$\frac{\partial\Phi}{\partial t} = V(\kappa)\,|\nabla\Phi| , \tag{2.18}$$

where the initial function Φ_0 is given such that its zero level equals the desired initial contour.

The representation is now parametrisation free. At the initialisation stage the level set function Φ_0 has to be determined according to the initial contour C_0. Therefore a singed distance transformation is applied, and each pixel of Φ_0 will contain its distance to the zero level set.

We will not describe the actual numerical algorithms of the level set method. Standard implementations often suffer from problems like reinitialisation issues. In [55], this problem was addressed. The basic implementation is also described in [15]. More implementation and stability related issues are discussed in [30].

When applying level set methods to the GAC model with the Euler Lagrange equation given as in Equation 2.10 we get the following formulation:

$$\frac{\partial \Phi}{\partial t} = (\kappa + v_0)\, g\, |\nabla \Phi| + \nabla g \cdot \nabla \Phi \ . \tag{2.19}$$

Here v_0 is an arbitrary value that can shrink or expand the curve. The data term g will slow down the curve if it passes through a boundary. The term $\nabla g \cdot \nabla \Phi$ will even pull the contour back if it is near a boundary. As already mentioned the level set method has the advantage of allowing topological changes of the contour. Unfortunately the level set formulation only finds local minima. The initial contour has to be initialised close to the final one. Another drawback is the slow convergence time of the current implementations.

Level set methods can also be applied to the Active Contours Without Edges model [29, 56] that is discussed in Section 2.3.3.1.

2.2.3 Graph-Based Methods

Graph-based methods rely on the partitioning of a graph that is build depending on an underlying image. We will start by defining an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that contains a set of nodes $\mathcal{V}$ that correspond to the pixels or voxels, and a set of undirected edges $\mathcal{E}$. The edges are defined based on the chosen neighbourhood system. Additionally to the pixels there are other nodes, that are also called terminals, representing the background and foreground (or objects). See the two left images of Figure 2.2 for an illustration of the graph construction. To each edge $e \in \mathcal{E}$ a non-negative weight w_e is assigned, that is also called the cost.

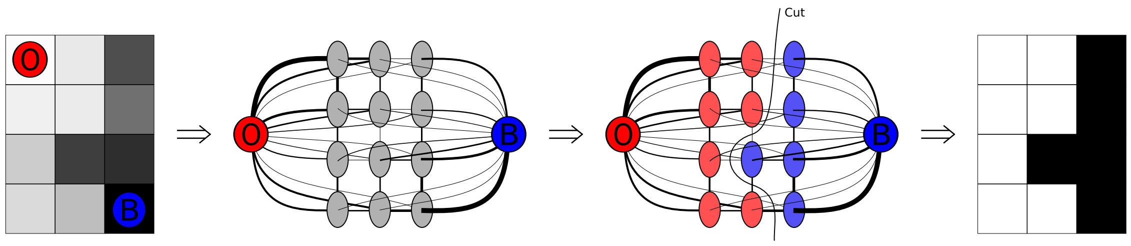


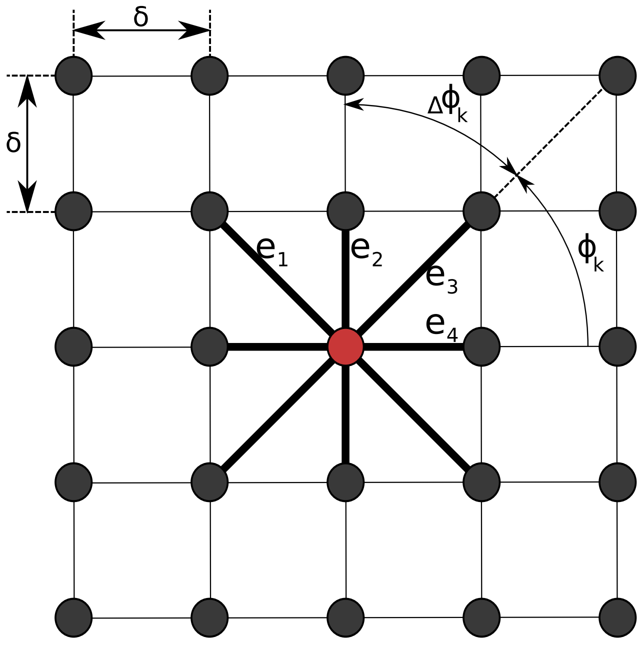
Figure 2.2: A graph is constructed based on an underlying image. The weights of the edges define the relationship between pixels. The graph cut algorithm finds a cut that separates the terminals with adherent nodes from each other, leading to the segmentation in image space.

2.2.3.1 Graph Cuts

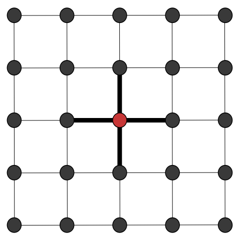
We will now describe the basic idea of graph cuts [11, 39] focusing on its application to compute geodesic minimal surfaces as described by Boykov et al. in [12]. A cut separates a graph around the terminals and represents a subset of edges $C \subset \mathcal{E}$. This is equal to the segmentation of the underlying image as can be seen in Figure 2.2. To each cut a cost is defined $|C| = \sum_{e \in C} w_e$ that sums up all the costs of the edges that the cut intersects. Thus the length of a cut can be defined as

$$|C|_{\mathcal{G}} = \sum_{e \in C} w_e \ . \tag{2.20}$$

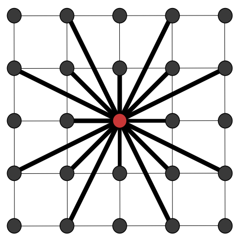
By suitably choosing the cost w_e we can approximate the Euclidean length of the contour in the image $|C|_{\mathcal{E}}$. Therefore we first have to make some notes on image grids. A regular 2D grid has a specific grid size δ and a defined neighbourhood system, as can be seen in Figure 2.3.



(a)



(b)



(c)

Figure 2.3: (a) 8-neighbourhood system. (b) 4-neighbourhood system. (c) 16-neighbourhood system.

A neighbourhood system is described by a set $E_{\mathcal{G}} = \{e_k : 1 \leq k \leq n_{\mathcal{G}}\}$ of vectors. An 8-neighbourhood system can be described by a set of four vectors $E_{\mathcal{G}} = \{e_1, e_2, e_3, e_4\}$ with corresponding angular orientation Φ_k.

Boykov et al. showed in [12] that with the right choice of w_e

$$|C|_{\mathcal{G}} \rightarrow |C|_{\mathcal{E}} \ , \tag{2.21}$$

as δ, $sup_k(\Delta\Phi_k)$ and $sup_k(e_k)$ get towards zero. Thus it is possible to find globally minimum geodesic contours with an arbitrary Riemannian metric. In [12], the edge weights at a given point p were chosen as

$$w_k(p) = \frac{\delta^2 \left|e_k\right|^2 \Delta\Phi_k \mathbf{det} D(p)}{2\left(e_k^T D(p) e_k\right)^{3/2}} \ , \tag{2.22}$$

with an anisotropic Riemmannian metric that is derived from the image I:

$$D(p) = g\left(|\nabla I|\right) \boldsymbol{I} + \left(1 - g\left(|\nabla I|\right)\right) \boldsymbol{u}\boldsymbol{u}^T \ . \tag{2.23}$$

Where $\boldsymbol{u} = \frac{\nabla I}{|\nabla I|}$ is the normalised gradient vector of the image, and $\boldsymbol{I}$ the unit matrix. To solve the min-cut problem, they used a max-flow algorithm from [13]. Unfortunately to hold Equation 2.21 the neighbourhood system has to be very large, affecting performance. If the neighbourhood system is too small, metrication errors are introduced.

2.2.3.2 Random Walker

In [44], Grady compared the above algorithm to the random walker algorithm [43] and the isoperimetric algorithm [42], that is basically the same as the random walker but needs only foreground labels and no background seeds. In [80], Sinop and Grady present an algorithm that unifies the graph cut and random walker algorithm, that still can be derived as special cases of the algorithm. The new algorithm uses a *max* norm in the energy that has to be minimised. He showed that standard graph cuts suffer from the "small cut" problem, where the segmentation can result in the trivial segmentation given just by the seed points. The random walker does not suffer from this problem.

The basic idea of the random walker algorithm is to start with a random walker at each unlabelled pixel, and then find the probability that it first reaches each of the seed points. For the segmentation each pixel is assigned the most probable seed destination. This algorithm can be easily computed analytically, as the probability of a random walker first reaching a seed point equals exactly the solution of the Dirichlet problem [43]. The Dirichlet energy can be defined as

$$D[u] = \frac{1}{2} \int_{\Omega} |\nabla u|^2 \ . \tag{2.24}$$

Grady showed how to set up the graph correctly and derived a sparse linear equation system that has to be solved. This system can be solved directly by a LU decomposition which unfortunately requires lots of memory, and is therefore limited to small volumes. Alternatively iterative methods like the multigrid method or conjugate gradient algorithm can be applied. The quadratically norm of the random walker algorithm does actually not correspond to the true length of the contour.

As already mentioned the definition of the grid is of great importance for graph based methods. The metrication errors induced by a small neighbourhood system can be quite significant. Figure 2.4(a) shows the degraded segmentation using only a 4-neighbourhood system. On the other hand Figure 2.4(b) depicts the same segmentation with the algorithm proposed in this master's thesis. The light red areas represent the foreground seeds, and the blue border shows the background seeds.

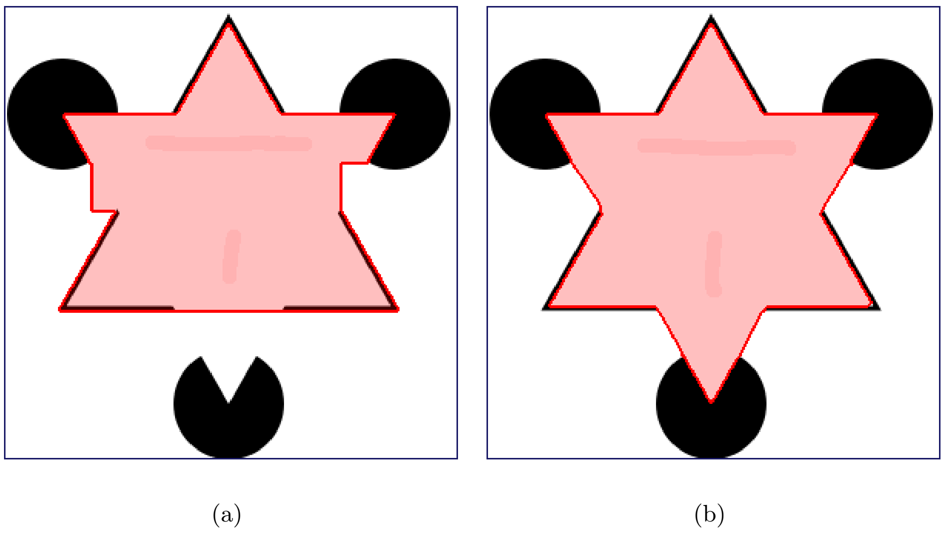


Figure 2.4: Segmentation using (a) 4-neighbourhood and (b) our approach.

2.2.4 Continuous Maximal Flow

In [4], Appleton and Talbot presented an approach to compute minimal surfaces using continuous maximal flows. The continuous maximal flow system is based on globally optimal geodesic active contours, and is defined as following:

$$\frac{\partial P}{\partial t} = -\nabla \cdot \boldsymbol{F} \, , \tag{2.25}$$

$$\frac{\partial \mathbf{F}}{\partial t} = -\nabla P \, , \tag{2.26}$$

subject to

$$|\mathbf{F}| \leq g \, . \tag{2.27}$$

P is a scalar potential field and $\mathbf{F}$ is a vector flow field, that is evolving over time. The foreground can be defined as source by $P(x) = 1$ and the background as a sink by $P(x) = 0$. With Equation 2.25 the potential P is updated according to the flow field $\mathbf{F}$. Equation 2.26 makes the flow dependent on the gradients in P. Together they form a system of wave equations. The constraint in Equation 2.27 on the magnitude of $\mathbf{F}$ regulates the diffusion process according to the Riemannian metric g. At strong borders the propagation is slowed down, while in flat regions evolution is almost unopposed. The algorithm is implemented as an iterative scheme with an artificial time step $\Delta t < \frac{1}{\sqrt{D}}$ for D-dimensional images. The final segmentation is obtained as the 0.5 level set of P.

In Section 3.1.3 we will see that the continuous maximal flow is closely related to the segmentation model proposed in this master's thesis.

2.3 The Total Variation Approach

Our approach is a variational one that uses the Total Variation (TV) of an image. The TV norm of an image is defined as

$$TV(u) = \int_{\Omega} |\nabla u| \, d\Omega \, , \tag{2.28}$$

with $|\nabla u| = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2}$ in two dimensions.

In the following we will review previous work that will lead us to our segmentation problem. A lot of work to apply TV methods to different computer vision tasks has been done recently [27]. Denoising algorithms and their solution are closely related to our approach of image segmentation. Therefore we will review them here first.

2.3.1 ROF Model

The first and most influential work using Total Variation in image processing was done by Rudin, Osher and Fatemi (ROF) in [76]. They presented an edge preserving image denoising model that was originally defined as

$$\min_u \left\{ \int_\Omega |\nabla u| \, d\Omega \right\} \, , \tag{2.29}$$

subject to

$$\int_\Omega (u - f)^2 \, d\Omega = \sigma^2 \, , \tag{2.30}$$

where the unknown u is the denoised image, and f is the observed image that is assumed to be corrupted by Gaussian noise with zero mean and standard deviation σ. The constraint makes sure that u is related to the observed image.

As shown in [25], the non-convex problem given by Equation 2.29 and 2.30 can be transformed into the following convex problem:

$$\min_u \left\{ E_{ROF} = \int_\Omega |\nabla u| \, d\Omega + \frac{1}{2\lambda} \int_\Omega (u - f)^2 \, d\Omega \right\} , \tag{2.31}$$

where $\lambda > 0$ is a Lagrange multiplier that has to be determined to solve Equations 2.29 and 2.30. λ is closely related to the scale of details in the recovered image. The ROF model is sometimes also referred to as the TV-L^2 model.

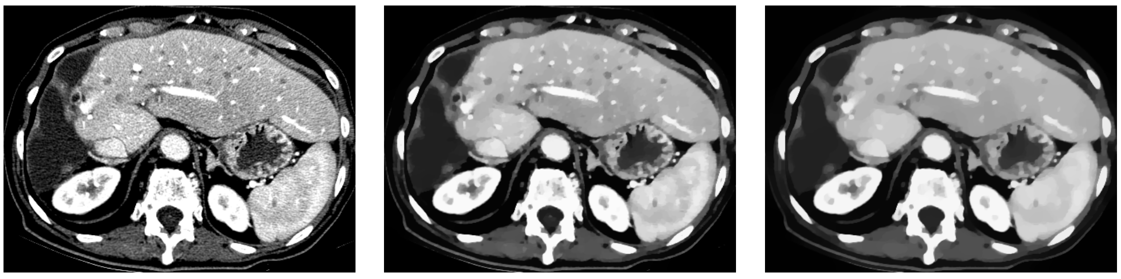

(a) (b) (c)

Figure 2.5: (a) Original test image. The ROF model applied with (b) $\lambda = 1$ and (c) $\lambda = 0.2$.

In Figure 2.5 the ROF denoising model is applied to a medical image. One can see how small features of the image disappear with decreasing λ. Strong edges are preserved while weak ones vanish. Unfortunately the effect of λ depends on the scale of the gray values, and is thus not contrast invariant. Moreover there is always some loss of contrast between the background and the remaining foreground, as can be seen in Figure 2.6(b,c).

2.3.2 L^1 Data Fidelity Term

By replacing the L^2 data fidelity term of Equation 2.31 with a L^1 term we arrive at the TV-L^1 model [7, 26, 65], that is defined as the following variational problem:

$$\min_u \left\{ E_{L^1} = \int_\Omega |\nabla u| \, d\Omega + \lambda \int_\Omega |u - f| \, d\Omega \right\} \; . \tag{2.32}$$

Although the changes are minor the effects are very important. The greatest improvement is that the algorithm has become contrast invariant, meaning that image structures of the same size, but different contrast to the background will vanish with the same value of λ. This effect can be seen in Figure 2.6, where the ROF and the TV-L^1 model are compared.

Furthermore the TV-L^1 model does not suffer from any contrast loss, and is also very good at removing impulse noise. In contrary to the ROF model the energy is not strictly convex, resulting in more than one global minimum.

The TV-L^1 model is perfectly suited to remove noise, select features of a certain scale and extract the texture from images. What makes the TV-L^1 model interesting for our task can be seen by its application to shape denoising.

For two shapes Σ and Ω that are defined on a subset of $\mathbb{R}^n$ and map to a binary set $[0, 1]$ we will now consider the following shape denoising problem:

$$\min_\Sigma \left\{ \operatorname{Per}(\Sigma) + \lambda \, |\Sigma \Delta \Omega| \right\} \; . \tag{2.33}$$

Here $\operatorname{Per}(\cdot)$ denotes the perimeter and represents here a regularisation term. The second term measures the symmetric difference between the two shapes Σ and Ω and is a data fidelity term. λ is once more a positive parameter that defines the balance between the two terms.

As the shapes can also be defined by their boundaries this problem is usually solved through a curve evolution process. The curve is iteratively deformed according to its underlying variational energy. Though level set methods as described in chapter 2.2.2 are quite commonly used for such a purpose, they tend to get stuck in local minima. This problem can be overcome by reformulating the model in Equation 2.33 in terms of the TV-L^1 model, as proposed by Nikolova et al. [66]. By rewriting the problem in terms of level sets, they showed that the non-convex shape denoising problem can be turned

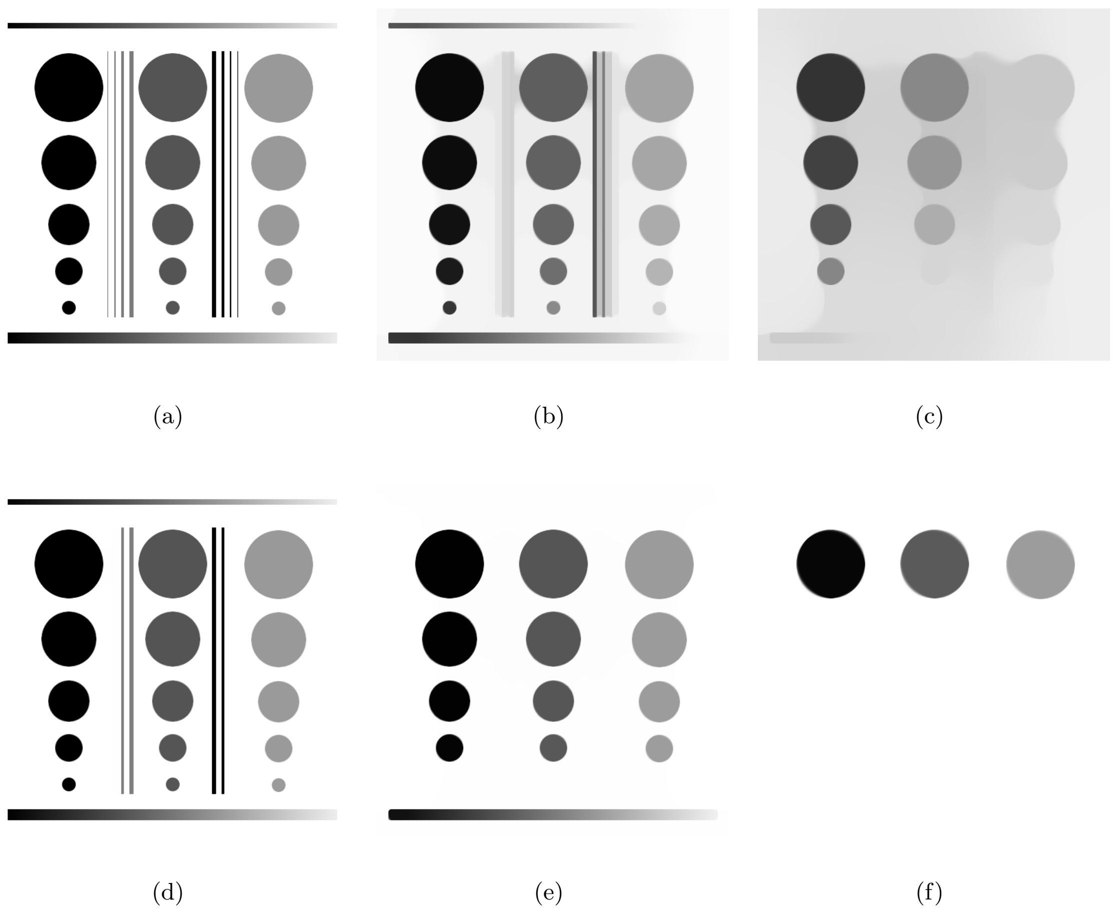


Figure 2.6: (a) Original test image. The ROF model applied with (b) $\lambda = 1$ and (c) $\lambda = 0.2$. The TV-L^1 model applied with (d) $\lambda = 1$, (b) $\lambda = 0.2$ and (c) $\lambda = 0.04$.

into an equivalent convex minimisation problem. Nikolova et al. proved that if $u(x)$ is a minimiser of Equation 2.32 for a binary input image $f(x) = \mathbf{1}_\Omega(x)$, then for almost every $\mu \in [0, 1]$ the set

$$\Sigma(\mu) = \{x \in \mathbb{R}^n : u(x) > \mu\} \ . \tag{2.34}$$

is also a minimiser of the shape denoising problem given in Equation 2.33.

Thus we can easily employ the TV-L^1 algorithm on a binary input image for the purpose of shape denoising. In Figure 2.7 the effect of different values for the parameter λ are shown. One can see that a small value of λ strongly penalises the length of the border. As already mentioned earlier, the TV-L^1 model possesses more than one global minimum. One can see in Figure 2.7 that the output images are not binary any more. By choosing a threshold value in the interval $[0,1]$ on can obtain all valid solutions of the minimisation problem.

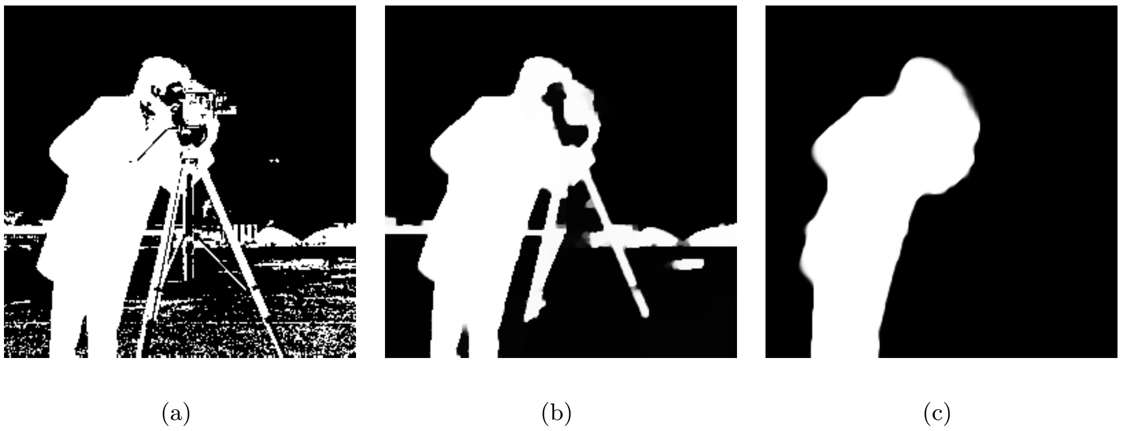


Figure 2.7: Shape denoising applied to a binary image (a) original shape. (b) TV-L^1 with $\lambda = 0.5$. (c) TV-L^1 with $\lambda = 0.1$.

2.3.3 Application to Segmentation

In the following we will review previous segmentation approaches based on the Total Variation. We will first review the Mumford-Shah functional that was widely used in the past. Furthermore we review different approaches, that are closely related to the proposed segmentation method, using the weighted Total Variation.

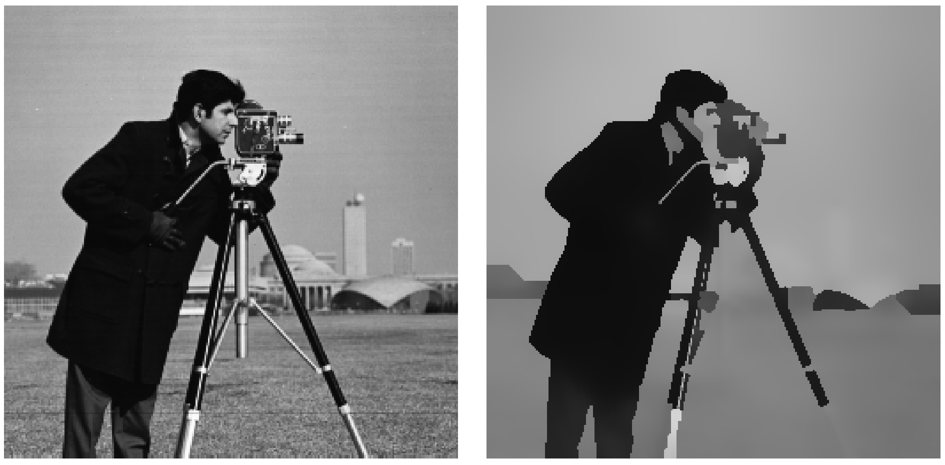
2.3.3.1 The Mumford-Shah Functional

The Mumford-Shah (MS) segmentation model [63, 64] determines the optimal piecewise smooth approximation of an image. It was one of the first segmentation models based on the Total Variation. The Mumford-Shah functional is defined as

$$E_{MS} = \int_{\Omega} |u - f|^2 \, d\Omega + \alpha \int_{\Omega \setminus \Gamma} |\nabla u|^2 \, d\Omega + \nu \text{length}(\Gamma) \, , \tag{2.35}$$

where f is the observed image that is defined on a domain Ω, u is the piecewise smooth approximation to f, Γ is a set of discontinuities that represent the edges of u, α and ν are arbitrary parameters. The first term makes sure that u is related to the observed image, and is therefore also called fidelity term. The second term is responsible for the regions to be smooth in the region $\Omega \setminus \Gamma$. Finally the last term is a regularisation term on the discontinuities.

The MS model cannot capture textured objects as the underlying image model assumes that the regions have homogeneous gray values, and sharp borders. An example of the MS model can be seen in Figure 2.8. One can note the piecewise constant areas of the MS segmentation, where each gray value corresponds to an object.



(a) (b)

Figure 2.8: (a) Original image and (b) the Mumford-Shah model applied to the image.

Another drawback of the MS model is the fact that its solution is not easy to realise, and the problem has to be reformulated to find a solution. Even then the solution is in general not unique. In [29], Chan and Vese presented the Active Contour Without Edges (ACWE) model that represents a special case of the MS model. They made a connection to active contours, and approximated the last term in Equation 2.35, by the length of a set of curves. In [29] the ACWE model was solved using level set methods.

2.3.3.2 The g-weighted Total Variation

In [15–17], Bresson et al. introduced the g-weighted Total Variation norm

$$TV_g(u) = \int_\Omega g(x)|\nabla u| d\Omega \; . \tag{2.36}$$

He showed that if u is a characteristic function $\mathbf{1}_C$, Equation 2.36 is equivalent to Equation 2.8. In other words, the g-weighted Total Variation model is equal to the GAC segmentation model. Note that the characteristic function $\mathbf{1}_C$ is a closed set in the image domain Ω and C stands for its boundary. The promise of this formulation is that if u is allowed to vary smoothly between $[0, 1]$, Equation 2.36 becomes a convex functional, meaning that one can compute the global minimiser of it. The final segmentation can in turn be extracted from u by selecting a level set in $[0, 1]$.

As already mentioned for the GAC model, the trivial solution $u = \mathit{const}$ is always a global minimiser of Equation 2.36. To account for this, one has to restrict the space of possible solutions by incorporating some constraints. In [16], Bresson et al. coupled the minimisation of Equation 2.36 with the Mumford Shah functional [29]. This effectively prevents the GAC model from yielding a trivial solution but on the other hand reduces the flexibility of the model.

In [17], Bresson et al. also used the weighted TV norm together with the TV-L^1 data fidelity term. When u is the characteristic function $\mathbf{1}_C$ the TV-L^1 minimisation problem (as in Equation 2.32) with the weighted TV norm becomes

$$\min_u \left\{ E_B = \int_\Omega g(x) \left|\nabla \mathbf{1}_C\right| d\Omega + \lambda \int_\Omega \left|\mathbf{1}_C - f\right| d\Omega \right\} . \tag{2.37}$$

As $\int_\Omega g(x) \left|\nabla \mathbf{1}_C\right| d\Omega = \int_C g(x) ds = E_{GAC}(C)$ the minimisation of the energy E_B is equal to minimise the GAC energy defined in Equation 2.8, with the image f being approximated by a binary function $\mathbf{1}_C$. The energy in Equation 2.37 is not strictly convex, meaning that more than one global minimum may exist. As we will see later, the energy E_B is closely related to our proposed segmentation model.

Another closely related work was done by Leung and Osher. In [54], they unified image denoising, segmentation and inpainting. The idea is to use Equation 2.36 together with a spatially varying L^1 data fidelity term:

$$\min_u \left\{ E_L = \int_\Omega g(x) \left|\nabla u\right| d\Omega + \int_\Omega \lambda(x) \left|u - f\right| d\Omega \right\} . \tag{2.38}$$

Although they used the same energy functional as proposed in this master's thesis, they used it mainly for denoising rather than segmentation. Instead they focused on the application to image denoising and inpainting. If $\lambda = 0$ the algorithm performs inpainting, as the energy does not depend on the observed data. In Figure 2.9 an example of gray value inpainting using a slightly modified version of our TV-L^1 implementation is given. In Figure 2.9(b) the original image is corrupted with 75% Salt-and-Pepper noise. As one can see in Figure 2.9(c) that even simple TV-L^1 denoising can significantly improve the

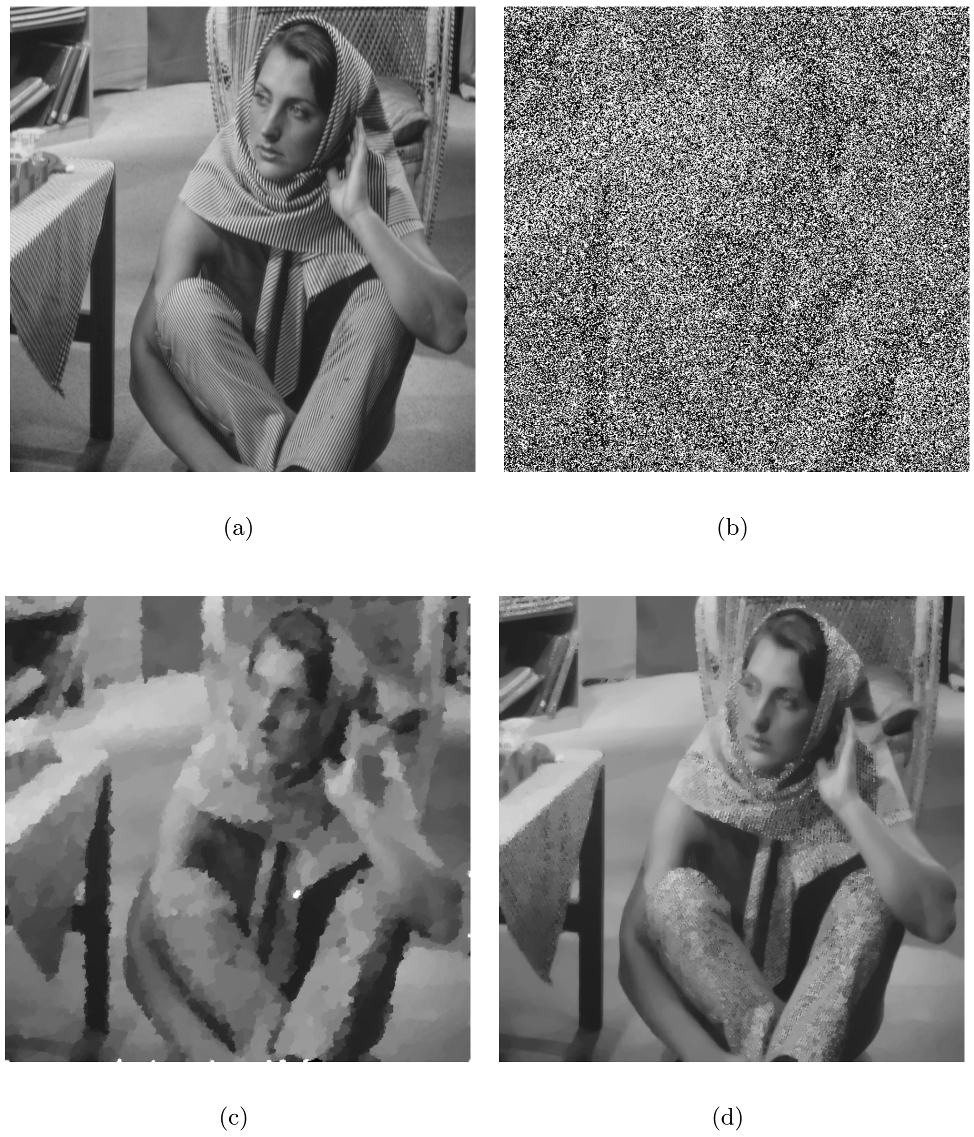


Figure 2.9: Gray value inpainting on *Barbara*: (a) the original image, (b) the image corrupted with 75% Salt-and-Pepper noise, (c) denoising using the TV-L^1 model and (d) inpainting on the gray values 0 and 255.

image. Salt-and-Pepper noise has the advantage that it is easy to detect. Therefore it is simple to use inpainting only at noisy pixels. In Figure 2.9(d) the result using inpainting to remove Salt-and-Pepper noise is shown.

Instead of coupling the minimisation of the GAC model with some unsupervised data driven constraints as done by Leung and Osher, we propose to include local constraints provided by the user. We will discuss our segmentation model in detail in Section 3.

2.4 Solving Total Variation Models

In Section 2.3 we presented different Total Variation denoising functionals. The fast minimisation of these energies is an important part of this master's thesis. In the following we will derive and discuss different approaches that are used for minimisation. We note that all of the following approaches are well suited for parallel implementations.

2.4.1 Explicit Time Marching of the PDE

We will first consider the ROF model defined in Section 2.3.1. A common way to solve an energy functional is to solve the associated Euler-Lagrange equation. For the ROF model defined in Equation 2.31 the Euler-Lagrange equation is given by

$$-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) + \frac{1}{\lambda} (u - f) = 0 \; . \tag{2.39}$$

One can notice that the equation is not defined for $\nabla u = \mathbf{0}$. A commonly used approach to overcome this problem is to replace $|\nabla u|$ by a regularised version $|\nabla u|_\epsilon = \sqrt{|\nabla u|^2 + \epsilon}$. Unfortunately this solution leads to blurring of the result for high values of ϵ. It should also be remarked that Equation 2.39 is highly non-linear.

To solve the PDE equation Rudin, Osher and Fatemi proposed in [76] an explicit time marching scheme. By introducing an artificial time step, we arrive at

$$u^{n+1} = u^n - dt \left[-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|_\epsilon} \right) + \frac{1}{\lambda} (u - f) \right] \; . \tag{2.40}$$

This approach represents a steepest descend method, and is iterated until a steady state is reached. A great drawback of the explicit time marching scheme is that it is very slow, especially when $|\nabla u|$ is small.

2.4.2 Primal Formulation

Another approach was introduced by Vogel and Oman in [89]. They proposed a fixed point algorithm that linearises Equation 2.39 by taking the non-linear term $|\nabla u|_\epsilon$ from the previous iteration. At each iteration $n + 1$, the following sparse system of linear equations has to be solved:

$$-\nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|_\epsilon} \right) + \frac{1}{\lambda} \left(u^{n+1} - f \right) = 0 \; . \tag{2.41}$$

To solve this equation system, a Jacobi or Gauss-Seidel algorithm or any other sparse solver can be used. Practise showed that a few, or even a single, Jacobi iteration is sufficient to achieve convergence of the entire algorithm, although single iterations are not solved exactly. The fixed point algorithm converges much faster than the explicit time marching scheme. Unfortunately this approach still suffers from the regularisation parameter ϵ, as the edge preserving capabilities of the ROF model are lost for great values of ϵ. On the other hand the algorithm gets very slow for small values of ϵ.

We now want to find a solution of the TV-L^1 model as defined in Equation 2.32. The according Euler-Lagrange equation is given as

$$-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) + \lambda \frac{(u-f)}{|u-f|} = 0 \ . \tag{2.42}$$

This equation is not defined for $\nabla u = \mathbf{0}$ and for $u - f = 0$. This PDE is even more degenerated than the ROF model, but one can use the same approach to overcome this problem here too. We therefore simply replace $|\nabla u|$ with $|\nabla u|_\epsilon = \sqrt{|\nabla u|^2 + \epsilon}$ and additionally we replace $|u - f|$ with $|u - f|_\delta = \sqrt{|u - f|^2 + \delta}$.

It is now possible to solve the TV-L^1 model the same way as the ROF model using the fixed point algorithm of Vogel and Oman. For each iteration $n + 1$ the following sparse equation system has to be solved:

$$-\nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|_\epsilon} \right) + \lambda \frac{(u^{n+1}-f)}{|u^n-f|_\delta} = 0 \ . \tag{2.43}$$

As it is also the case for the ROF algorithm, the TV-L^1 algorithm is very sensitive to the choice of the parameters ϵ and δ.

We implemented the fixed point algorithm for the ROF and TV-L^1 model to evaluate the speed of the different approaches. We will also refer to the fixed point algorithm as the primal approach.

2.4.3 Dual Formulation

The primal formulation in the previous section suffered from a degeneration of the PDE as $u \to 0$. To overcome this problem, the dual formulation was studied by Chambolle [23], Chan et al. [28] and Carter et al. [20].

2.4.3.1 Chambolle's Algorithm

In the following we will present an alternative derivation of the Chambolle algorithm presented in [23] for the ROF model. The basic idea is to replace the problematic term $\frac{\nabla u}{|\nabla u|}$ in the Euler-Lagrange equation defined in Equation 2.39 with the vector $\boldsymbol{p}$:

$$-\nabla \cdot \underbrace{\left(\frac{\nabla u}{|\nabla u|}\right)}_{\boldsymbol{p}} + \frac{1}{\lambda}(u-f) = 0 \,. \tag{2.44}$$

Thus $\boldsymbol{p}$ should be given as

$$\boldsymbol{p}(x) = \begin{cases} \frac{\nabla u(x)}{|\nabla u(x)|} & \text{if} \quad \nabla u(x) \neq 0 \\ \text{not unique} & \text{if} \quad \nabla u(x) = 0 \end{cases} \tag{2.45}$$

In Figure 2.10 this relation is illustrated.

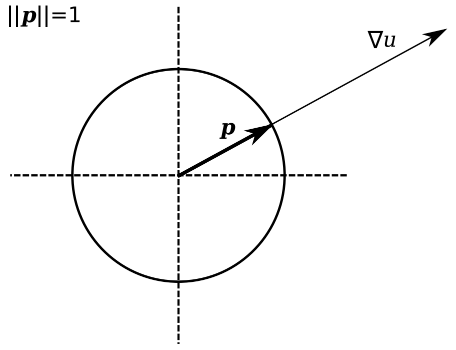


Figure 2.10: Illustration of the dual variable $\boldsymbol{p}$ that is the normalised gradient ∇u.

Using the primal-dual formulation in Equation 2.44, we obtain the following two equations:

$$\begin{aligned} -\nabla \cdot \boldsymbol{p} + \frac{1}{\lambda}(u-f) = 0 \,, \\ \boldsymbol{p}\,|\nabla u| - \nabla u = 0 \,. \end{aligned} \tag{2.46}$$

Now we can apply an iterative fixed point algorithm (or gradient descend) to the second equation:

$$\boldsymbol{p}^{n+1} = \boldsymbol{p}^{n} - dt\left(\boldsymbol{p}^{n+1}\,|\nabla u| - \nabla u\right) \,. \tag{2.47}$$

The time step is chosen to be $dt = \frac{\tau}{\lambda}$ with $\tau > 0$. When solving this equation for $\boldsymbol{p}^{n+1}$ we get:

$$\boldsymbol{p}^{n+1} = \frac{\boldsymbol{p}^{n} + (\tau/\lambda)\,\nabla u}{1 + (\tau/\lambda)\,|\nabla u|} \,. \tag{2.48}$$

This update scheme still depends on u. We can eliminate u by reformulating the first equation of Equation 2.46 as

$$u = \lambda \nabla \cdot \boldsymbol{p} + f \ . \tag{2.49}$$

Thus we arrive at our update scheme for the dual variable $\boldsymbol{p}$:

$$\boldsymbol{p}^{n+1} = \frac{\boldsymbol{p}^n + \tau \nabla \left(\nabla \cdot \boldsymbol{p}^n + f/\lambda \right)}{1 + \tau \left| \nabla \left(\nabla \cdot \boldsymbol{p}^n + f/\lambda \right) \right|} \ , \tag{2.50}$$

with $\boldsymbol{p}^0 = \mathbf{0}$. Chambolle proved in [23] that this algorithm converges as long as $\tau \leq 1/8$. Practise showed that the algorithm converges as long as $\tau \leq 1/4$. As convergence speed increases with increasing τ, we chose $\tau = 1/4$ for our implementations in 2D. The desired solution u can be recovered from the dual variable at any time by solving Equation 2.49.

2.4.3.2 Projected Gradient Descend

In the following we show an alternative algorithm to solve the ROF model based on the considerations made in [20, 24, 28]. Similar as in the previous section we want the dual variable $\boldsymbol{p}$ to be the 1-normalised image gradient u as illustrated in Figure 2.10. The desired properties of $\boldsymbol{p}$ can be achieved by maximising the in-product of $\boldsymbol{p}$ and ∇u:

$$|\nabla u| = \max_{\|\boldsymbol{p}\| \leq 1} \left\{ \boldsymbol{p} \cdot \nabla u \right\} \ . \tag{2.51}$$

On can see that when $\|\boldsymbol{p}\| = 1$ we get in two dimensions

$$p_1 = \frac{u_x}{\sqrt{u_x^2 + u_y^2}} \quad \text{and} \quad p_2 = \frac{u_y}{\sqrt{u_x^2 + u_y^2}} \ . \tag{2.52}$$

To show the correctness of Equation 2.51 we can now write

$$\begin{aligned} \boldsymbol{p} \cdot \nabla u &= p_1 u_x + p_2 u_y \\ &= \frac{u_x^2}{\sqrt{u_x^2 + u_y^2}} + \frac{u_y^2}{\sqrt{u_x^2 + u_y^2}} \\ &= \sqrt{u_x^2 + u_y^2} = |\nabla u| \ . \end{aligned} \tag{2.53}$$

By substituting Equation 2.51 into the original formulation of the ROF model as defined in Equation 2.31, we get the primal-dual formulation of the ROF model as

$$\min_{u} \max_{\|\boldsymbol{p}\|\leq 1} \left\{ \int_{\Omega} \boldsymbol{p} \cdot \nabla u \, d\Omega + \frac{1}{2\lambda} \int_{\Omega} (u - f)^2 \, d\Omega \right\} . \tag{2.54}$$

Here we can easily interchange max and min, as our problem is convex in u. One can now derive this equation by using the divergence theorem, stating that in a continuous setting $-\int u \nabla \cdot \boldsymbol{p} = \int \boldsymbol{p} \cdot \nabla u$. Setting the first derivation zero we obtain

$$-\nabla \cdot \boldsymbol{p} + \frac{1}{\lambda} (u - f) = 0 . \tag{2.55}$$

This is the same optimality condition we already derived in Equation 2.49. Using this condition we can now easily eliminate u from Equation 2.54:

$$\max_{\|\boldsymbol{p}\|\leq 1} \left\{ -\int_{\Omega} \lambda (\nabla \cdot \boldsymbol{p})^2 \, d\Omega - \int_{\Omega} f \nabla \cdot \boldsymbol{p} \, d\Omega + \frac{1}{2\lambda} \int_{\Omega} \lambda^2 (\nabla \cdot \boldsymbol{p})^2 \, d\Omega \right\} . \tag{2.56}$$

By simplifying and reformulating this equation, we finally arrive at the dual formulation of the ROF model, that is given as the optimisation problem

$$\min_{\|\boldsymbol{p}\|\leq 1} \left\{ \int_{\Omega} f \nabla \cdot \boldsymbol{p} \, d\Omega + \frac{\lambda}{2} \int_{\Omega} (\nabla \cdot \boldsymbol{p})^2 \, d\Omega \right\} . \tag{2.57}$$

The main advantage of this dual formulation is, that it is continuously differentiable. Therefore no approximations as in the primal algorithm are necessary. The Euler-Lagrange equation associated to the dual formulation of the ROF model is given as

$$-\nabla (f + \lambda \nabla \cdot \boldsymbol{p}) = 0 , \quad \|\boldsymbol{p}\| \leq 1 . \tag{2.58}$$

In the dual formulation we have to account for the additional constraint $\|\boldsymbol{p}\| \leq 1$. Nevertheless it is possible to calculate the solution by a simple gradient descent, and then re-project the intermediate dual variable $\tilde{\boldsymbol{p}}$ in a second step. Thus the projected gradient descend algorithm [24] can be written as

$$\begin{aligned} \tilde{\boldsymbol{p}}^{n+1} &= \boldsymbol{p}^n + (\tau/\lambda) \left[\nabla (f + \lambda \nabla \cdot \boldsymbol{p}^n) \right] \\ \boldsymbol{p}^{n+1} &= \frac{\tilde{\boldsymbol{p}}^{n+1}}{\max \{1, |\tilde{\boldsymbol{p}}^{n+1}|\}} . \end{aligned} \tag{2.59}$$

This algorithm is fast and robust, and provides an exact solution of the convex ROF model. Experiments in Section 5.2 showed that the projected gradient descend algorithm converges faster than Chambolle's algorithm described in the last section.

2.4.3.3 Dual TV-L^1 Algorithm

Solutions to the TV-L^1 model were already discussed in [7, 26, 66]. As the TV-L^1 model defined in Equation 2.32 is not strictly convex, the principles used in the previous sections, cannot be applied directly. Therefore one wants to reformulate the problem to become strictly convex. Aujol et al. proposed in [7] the following convex approximation to the TV-L^1 model

$$\min_{u,v} \left\{ \int_\Omega |\nabla u| \, d\Omega + \frac{1}{2\theta} \int_\Omega (u-v)^2 \, d\Omega + \lambda \int_\Omega |v-f| \, d\Omega \right\} \ , \tag{2.60}$$

with $\theta > 0$. The problem has now become a convex optimisation problem of two variables. In the data fidelity term u is replaced by v. The newly introduced middle term ensures that v is close to u. One can note that if $\theta \to 0$ the model is exactly the TV-L^1 model, but if $\theta > 0$ then Equation 2.60 turns into a convex approximation. Practise showed that the algorithm is robust even for high values of θ.

To minimise the TV-L^1 energy approximation, a two step algorithm has to be performed, where in each step another variable is kept constant. When keeping v constant we obtain the following minimisation problem:

$$\min_{u} \left\{ \int_\Omega |\nabla u| \, d\Omega + \frac{1}{2\theta} \int_\Omega (u-v)^2 \, d\Omega \right\} \ . \tag{2.61}$$

This is exactly the ROF model as defined in Equation 2.31. The algorithms used in the previous sections can be used to solve for u.

In a second step, u is kept constant, resulting in the following minimisation problem:

$$\min_{v} \left\{ \frac{1}{2\theta} \int_\Omega (u-v)^2 \, d\Omega + \lambda \int_\Omega |v-f| \, d\Omega \right\} \ . \tag{2.62}$$

The associated Euler Lagrange equation is given as

$$\frac{1}{\theta}(v-u) + \lambda \frac{v-f}{|v-f|} = 0 \ , \tag{2.63}$$

where $\frac{v-f}{|v-f|}$ is the sign of $v-f$. To find a solution one has to consider the following three cases:

1. $v - f > 0$: Solving Equation 2.63 for v we get $v = u - \lambda\theta$.
2. $v - f < 0$: Similar to above we get $v = u + \lambda\theta$.
3. $v - f = 0$: Obviously, here is $v = f$.

By reformulating the conditions to be depending on u we get the following thresholding scheme for updates of v:

$$v = \begin{cases} u - \lambda\theta & \text{if} \quad u - f > \lambda\theta \\ u + \lambda\theta & \text{if} \quad u - f < -\lambda\theta \\ f & \text{if} \quad |u - f| \leq \lambda\theta \end{cases} \tag{2.64}$$

To find an appropriate value for θ we chose $\lambda\theta = \mathit{const}$. By adapting θ this way, the error made by the approximation is always the same, as we can see from Equation 2.64. We can also call $\lambda\theta$ a convexification threshold. Furthermore this choice guarantees fast convergence times.

We can summarise the algorithm as iterating the following two steps until convergence:

- Find a solution for u by applying a few Chambolle or projected gradient descend iterations.
- Then use the thresholding scheme of Equation 2.64 to update v.

Practise showed that the overall algorithm converges very fast even for a small number of iterations when solving for u. The dual approach converges much faster than the primal one discussed in Section 2.4.2, as one can also see by experiments in Section 5.2.

Chapter 3

Method

Contents

3.1 The Segmentation Model

In this section, we will introduce the used segmentation model, and discuss some of its properties. Methods to solve the minimisation problems as discussed in Section 2.4 are adapted to the segmentation problem, and the implemented algorithm is derived. Furthermore we will show how the proposed segmentation model is linked to the Maximal Flow algorithm of Appleton and Talbot.

3.1.1 Proposed Energy

Section 2.3.3 showed that the Total Variation can be used for image segmentation. We propose to minimise the following variational image segmentation model:

$$\min_{u\in[0,1]} \left\{ E_{seg} = \int_\Omega g(x)|\nabla u| d\Omega + \int_\Omega \lambda(x)\, |u - f|\, d\Omega \right\} . \tag{3.1}$$

The first term of the energy is exactly the g-weighted Total Variation of u as defined in Equation 2.36. The right hand term is a TV-L^1 data term with spatially varying $\lambda(x) > 0$. The function $f \in [0,1]$ is provided by the user and contains information about foreground ($f = 1$) and background ($f = 0$). This information can be obtained either by thresholding (that we will also refer to as weak constraints), or by manually setting local hard constraints. We mention that the constraint $u \in [0,1]$ can be omitted since $f \in [0,1]$.

The value of the spatially varying parameter $\lambda(x)$ has strong influence on the behaviour of the algorithm. We differ between three possible cases:

- $\lambda = 0$: In this case the right hand term of E_{seg} vanishes, and the information contained in f is no longer used. If $\lambda = 0$ for all $x \in \Omega$ then our formulation is exactly the weighted Total Variation as defined in Equation 2.36. Thus the pure globally optimal geodesic energy is computed.

- $\lambda \rightarrow \infty$: This results in hard constraints. The weighted Total Variation gets insignificant, and the right hand term of Equation 3.1 forces $u = f$.

- $0 < \lambda < \infty$: In this case the full energy E_{seg} is minimised. While we still minimise the GAC energy also the weak constraints in f get approximated. Thus shape denoising considering the geodesic energy, is applied to the initial seed regions obtained by thresholding. The smaller the value of λ the more smoothing is applied.

Now we can be more specific on the definition of hard constraints. For a foreground seed we have to set $\lambda = \infty$, and $f = 1$. A background seed is obtained with $\lambda = \infty$ and $f = 0$. Hard constraints will not be altered in any case, and are used to incorporate information from the user into the segmentation process. In addition we allow that the edge detection function $g(x)$ can be altered by the user, i.e. edges can be deleted or added in an interactive manner. Furthermore the user can set spatially different values for λ.

There exist two different approaches to obtain a segmentation with the proposed segmentation model:

- **Weak constraints**: Here the user has to select thresholds to initialise f with the seed regions. The obtained shape in f can be approximated (denoised) by selecting $0 < \lambda < \infty$. The user can spatially vary λ. Thus in different regions varying smoothing is applied to the initial contour. Hard constraints can be incorporated to further improve the segmentation.

- **Pure geodesic energy**: By setting $\lambda = 0$, geodesic active contours are computed. Our method always provides the global optimal solution. As already mentioned in Section 2.3.3 the trivial solution $u = const$ is always a global minimiser of our energy. Therefore the pure geodesic energy approach requires user interaction to set at least one foreground and one background constraint.

Of course both approaches can be combined throughout a single segmentation process. Practise proved that either one or the other way should be used as an optimal workflow to obtain the segmentation.

Note that u is no longer a characteristic function but can vary continuously between $[0, 1]$. As the energy in Equation 3.1 is not strictly convex, more than one global minimum exists. The final segmentation has to be chosen as a level set of u. Figure 3.1 shows an example of a simple segmentation using the pure geodesic energy approach. Two seed regions were set, as can be seen in Figure 3.1(a). The light red area represents foreground constraints, and the blue area indicates background constraints. In Figure 3.1(b) the edge information is depicted. Note that there exist borders of different strength. Figure 3.1(c) shows the variable u together with the constraints. As already mentioned, u is continuous. In Figure 3.1(d-f) possible global optimal solutions of the segmentation model are shown. Usually the solutions are very close together, and the $u = 0.5$ level set can be used for all segmentations.

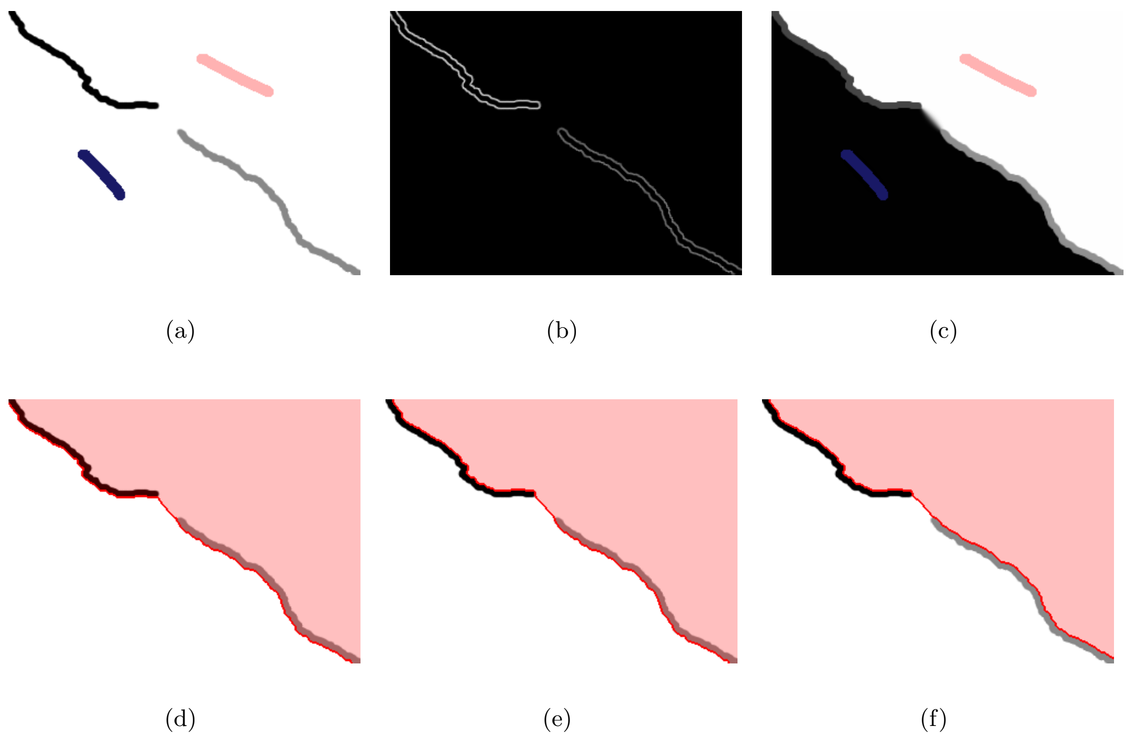


Figure 3.1: Segmentation example for an artificial image using the pure geodesic energy: (a) original image with foreground seed in red and background seed in blue, (b) the edge image and (c) the variable u. Different possible solutions are given for the level set at (d) 0.2, (e) 0.5 and (f) 0.8.

The pure geodesic energy approach relies solely on edge information. Therefore a good edge image is desired. In [48], Huang and Mumford proposed the following edge detection function for natural images:

$$g\left(|\nabla I|\right) = \exp\left(-\alpha\left|\nabla I\right|^{\beta}\right) \ . \tag{3.2}$$

The proposed parameter $\beta = 0.55$ proved well for all of our segmentation tasks.

3.1.2 Solving the Segmentation Model

We will now focus on the solution of the energy defined in Equation 3.1. Therefore we will consider the derivations we already made for the image denoising task in Section 2.4.3. The advantage of Chambolle's algorithm and the projected gradient descend algorithm, is that no ϵ-regularisation of the Total Variation term is needed. Unfortunately, this approaches can not be used with spatially varying parameter $\lambda(x)$. To account for this, we introduce an auxiliary variable v and propose to minimise the following approximation of E_{seg}

$$\min_{u,v}\left\{\int_{\Omega} g(x)|\nabla u|d\Omega + \frac{1}{2\theta}\int_{\Omega}(u-v)^2\,d\Omega + \int_{\Omega}\lambda(x)\,|v-f|\,d\Omega\right\} \ . \tag{3.3}$$

We first note that as $\theta \to 0$, Equation 3.3 approaches Equation 3.1. Moreover we note that Equation 3.3 is still convex. This means that we can compute the global minimiser of it. However, unlike E_{seg}, the new energy is now an optimisation problem in two variables, u and v. Therefore we have to perform an alternating minimisation as already done in Section 2.4.3.3.

In the first step we have to solve Equation 3.3 for u, with a fixed v. The resulting energy functional to be minimised is now

$$\min_{u}\left\{\int_{\Omega} g(x)|\nabla u|d\Omega + \frac{1}{2\theta}\int_{\Omega}(u-v)^2\,d\Omega\right\} \ . \tag{3.4}$$

The according Euler Lagrange equation is given as

$$-\nabla\cdot\left(g(x)\frac{\nabla u}{|\nabla u|}\right) + \frac{1}{\theta}\left(u-v\right) = 0 \ . \tag{3.5}$$

We can see that this optimisation problem is exactly the ROF model, where θ is now a spatially constant regularisation parameter. The only difference to the original ROF model lies in the g-weighting of the Total Variation norm. The dual variable thus becomes

$$\boldsymbol{p} = g(x)\frac{\nabla u}{|\nabla u|} \ . \tag{3.6}$$

As a consequence we get the following Chambolle update scheme for the dual variable:

$$\boldsymbol{p}^{n+1} = \frac{\boldsymbol{p}^n + \frac{\tau}{\theta}\nabla u^n}{1 + \frac{\tau}{\theta}\frac{|\nabla u^n|}{g(x)}} \ , \tag{3.7}$$

with u defined as

$$u^n = \theta\nabla \cdot \boldsymbol{p}^n + v \ . \tag{3.8}$$

We already mentioned earlier that the projected gradient descend updates lead to an overall faster convergence of the algorithm. Therefore it is desirable to use the projected gradient descend algorithm for the minimisation of the segmentation model. When looking at the new definition of the dual variable $\boldsymbol{p}$ in Equation 3.6, we can note that $||\boldsymbol{p}|| = g(x) \leq 1$. To account for this modification the second equation in Equation 2.59 can simply be modified to

$$\boldsymbol{p}^{n+1} = \frac{\tilde{\boldsymbol{p}}^{n+1}}{\max\left\{1, \frac{|\tilde{\boldsymbol{p}}^{n+1}|}{g(x)}\right\}} \ . \tag{3.9}$$

With this modification the projected gradient descend algorithm can be used to minimise the energy in Equation 3.4.

Now we have to solve the model for v, while u is fixed. The resulting minimisation problem is given as

$$\min_v \left\{\frac{1}{2\theta}\int_\Omega (u-v)^2\, d\Omega + \int_\Omega \lambda(x)\,|v-f|\, d\Omega\right\} \ , \tag{3.10}$$

with the corresponding Euler Lagrange equation

$$-\frac{1}{\theta}(u-v) + \lambda(x)\frac{v-f}{|v-f|} = 0 \ . \tag{3.11}$$

One can immediately see that the minimisation problem in Equation 3.10 is similar to the minimisation problem in Equation 2.62. The only difference is the spatially varying parameter $\lambda(x)$. Nevertheless the same thresholding scheme as in Equation 2.64 can be used to solve for v.

In the following we summarise the proposed segmentation algorithm. The outline of the alternating minimisation procedure is as follows:

1. For fixed v, solve equation 3.3 for u: The projected gradient descend algorithm can be used as follows

$$\begin{aligned}\tilde{\mathbf{p}}^{n+1} &= \mathbf{p}^n + \frac{\tau}{\theta}\nabla u^n \\ \mathbf{p}^{n+1} &= \frac{\tilde{\mathbf{p}}^{n+1}}{\max\left\{1, \frac{|\tilde{\mathbf{p}}^{n+1}|}{g}\right\}} \\ u^{n+1} &= v^n + \theta\nabla\cdot\mathbf{p}^{n+1}\ .\end{aligned} \tag{3.12}$$

   Where we once more note that as in [24] for D-dimensional problems we have to chose $\tau \le \frac{1}{2D}$. We do not need to exactly solve this sub-optimisation-problem. One iteration of this scheme is sufficient to make the entire algorithm converge.

2. For fixed u, solve Equation 3.3 for v: The following thresholding scheme is used to update v:

$$v^{n+1} = \begin{cases} u^{n+1} - \lambda(x)\theta & \text{if} \quad u^{n+1} - f > \lambda(x)\theta \\ u^{n+1} + \lambda(x)\theta & \text{if} \quad u^{n+1} - f < -\lambda(x)\theta \\ f & \text{if} \quad \left|u^{n+1} - f\right| \le \lambda(x)\theta \end{cases} \tag{3.13}$$

3. Goto 1. until convergence.

The algorithm presented above is valid for an arbitrary value of $\lambda(x)$. The approximation parameter $\theta > 0$ was chosen as $\theta = 0.1$. The smaller the value of θ, the better our original functional is approximated. On the other hand θ can be seen as a kind of step size for the optimisation. Therefore the algorithm gets faster for higher values of θ.

We can once more take a look at the three possible cases for the parameter λ:

- $\lambda = 0$: Enables the algorithm to minimise the pure GAC energy as Equation 3.13 becomes $v = u$.

- $\lambda = \infty$: Accounts for fixed foreground or background seeds provided by the user, as Equation 3.13 becomes $v = f$

- $0 < \lambda < \infty$: Approximation of the weak constraints in f while minimising the geodesic active contour.

Figure 3.2 shows the evolution process of the variable u during the minimisation of the proposed segmentation model. Before running the minimisation algorithm the variable

u was initialised to $u = 0.5$ and the dual variable $\boldsymbol{p}$ was initialised to $\boldsymbol{p} = \mathbf{0}$. The approximation parameter θ was set to 0.5. Figure 3.2(a) shows the constraints as provided by the user. Foreground constraints set $u = f = 1$ and background constraints set $u = f = 0$. The border was set to be background, the foreground regions were drawn by the user. Specified regions in f correspond to $\lambda = \infty$, unspecified (grey) regions correspond to $\lambda = 0$. Figure 3.2(b) and Figure 3.2(c) show intermediate steps of the iterative minimisation algorithm, Figure 3.2(d) shows the final result of u. Once more one can see that more than one global optimal solution was found.

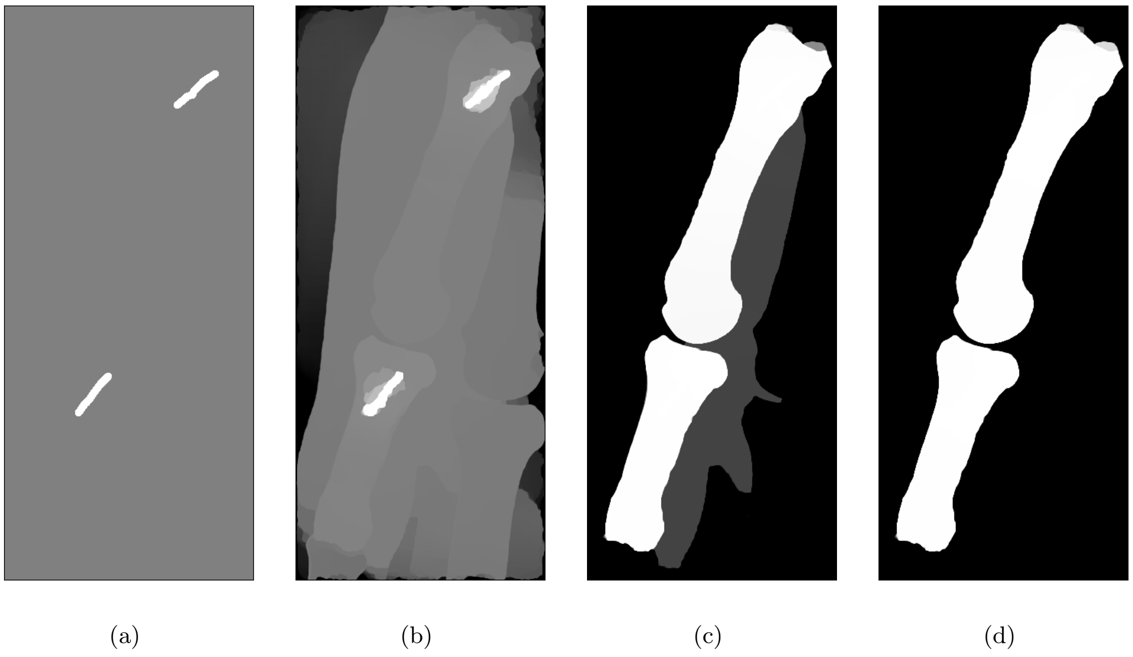


Figure 3.2: Evolution process of the variable u during minimisation of the proposed image segmentation model. (a) Initialisation, (b) intermediate step short after initialisation, (c) another intermediate step and (d) the final result.

3.1.3 Relation to Continuous Maximal Flows

We now show that in case of $\lambda = 0$ our algorithm is equivalent to the continuous maximal flow algorithm of Appleton and Talbot [4].

Let us first write down our algorithm for $\lambda = 0$, the second step of our algorithm is then given by $v = u$. Note that the data fidelity term in our proposed segmentation model (Equation 3.1) disappears. Therefore also a L^2 data fidelity term could have been used, as we proposed in [87]. We can therefore simplify our algorithm as

$$\begin{aligned}
\tilde{\mathbf{p}}^{n+1} &= \mathbf{p}^n + \frac{\tau}{\theta}\nabla u^n \\
\mathbf{p}^{n+1} &= \frac{\tilde{\mathbf{p}}^{n+1}}{\max\left\{1, \frac{|\tilde{\mathbf{p}}^{n+1}|}{g}\right\}} \\
u^{n+1} &= u^n + \theta\nabla\cdot\mathbf{p}^{n+1} \,. \qquad (3.14)
\end{aligned}$$

Based on Equation 2.25, Equation 2.26 and Equation 2.27 let us now write down the iterative algorithm of Appleton and Talbot:

$$\begin{aligned}
P^{n+1} &= P^n + \Delta t\nabla\cdot\mathbf{F}^n \\
\mathbf{F}^{n+1} &= \mathbf{F}^n + \Delta t\nabla P^{n+1} \\
|\mathbf{F}^{n+1}| &\leq g \,. \qquad (3.15)
\end{aligned}$$

It is now easy to see that for $\Delta t \equiv \theta \equiv \frac{\tau}{\theta}$ and up to an ordering of the updates both schemes are equivalent. Therefore, the continuous maximal flow algorithm of Appleton and Talbot essentially computes the minimiser of the weighted Total Variation functional. In other words, the projected gradient descend algorithm used in our segmentation algorithm, and first introduced in [24], is in principle equivalent to maximal flow algorithm of Appleton and Talbot.

3.2 Numerical Methods

In this section we derive the numerical methods used for implementation. All previous considerations were made in a continuous setting, and therefore we first have to make some considerations according the spatial discretisation.

3.2.1 Discretisation

On a digital computer the discretisation of the spatial image domain is required. A digital image can be seen as the sampling at discrete locations from a continuous image. Digital images are defined on a regular Cartesian grid. This implies that we work on a rectangular domain $\Omega = [x_1, x_m] \times [y_1, y_n]$ for two dimensional images, and on a domain $\Omega = [x_1, x_m] \times [y_1, y_n] \times [z_1, z_o]$ for three dimensional images. The discrete locations of the grid are given by

$$\begin{aligned}(x_i, y_j) &= (i\Delta x, j\Delta y) \mid 1 \leq i \leq m, 1 \leq j \leq n \\ (x_i, y_j, z_k) &= (i\Delta x, j\Delta y, k\Delta z) \mid 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq o\ ,\end{aligned} \tag{3.16}$$

respectively for two and three dimensions. Here Δx, Δy and Δz denote the spatial discretisation steps.

Of great importance is the discretisation of the derivative operators. We will here consider only two dimensions. In the discrete setting the gradient operator becomes:

$$\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)^T \quad \Longrightarrow \quad (\nabla u)_{i,j} = \left(\delta_x^+ u_{i,j}, \delta_y^+ u_{i,j}\right)^T\ . \tag{3.17}$$

Thus the Total Variation gets

$$|\nabla u| = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2} \quad \Longrightarrow \quad \left|(\nabla u)_{i,j}\right| = \sqrt{\left(\delta_x^+ u_{i,j}\right)^2 + \left(\delta_y^+ u_{i,j}\right)^2}\ . \tag{3.18}$$

The derivations are here approximated using forward differences, with the according border conditions, they are defined as

$$\begin{aligned}\delta_x^+ u_{i,j} &= \begin{cases} \frac{u_{i+1,j} - u_{i,j}}{\Delta x} & \text{if} \quad i < m \\ 0 & \text{if} \quad i = m \end{cases} \\ \delta_y^+ u_{i,j} &= \begin{cases} \frac{u_{i,j+1} - u_{i,j}}{\Delta y} & \text{if} \quad j < n \\ 0 & \text{if} \quad j = n\ . \end{cases}\end{aligned} \tag{3.19}$$

The approximation of the divergence operator can be done in a similar manner as

$$\nabla \cdot \boldsymbol{p} = \frac{\partial p^1}{\partial x} + \frac{\partial p^2}{\partial y} \quad \Longrightarrow \quad (\texttt{div}\, \boldsymbol{p})_{i,j} = \ \delta_x^- p_{i,j}^1 + \delta_y^- p_{i,j}^2\ , \tag{3.20}$$

where the backward differences are given by

$$\begin{aligned}\delta_x^- p_{i,j}^1 &= \begin{cases} \frac{p_{i,j}^1 - p_{i-1,j}^1}{\Delta x} & \text{if} \quad 1 < i < m \\ p_{i,j}^2 & \text{if} \quad i = 1 \end{cases} \\ \delta_y^- p_{i,j}^2 &= \begin{cases} \frac{p_{i,j}^2 - p_{i,j-1}^2}{\Delta y} & \text{if} \quad 1 < j < n \\ p_{i,j}^2 & \text{if} \quad j = 1 \qquad . \end{cases}\end{aligned} \tag{3.21}$$

For simplicity the algorithms in the next sections will be solely done for two dimensions.

As in all our 2D images $\Delta x = \Delta y = 1$ we can also simplify the derivation operators. We will not take into account boundary conditions when deriving the algorithms.

3.2.2 Primal Versions

We will now derive the numerical methods used to implement the fixed point algorithm of Section 2.4.2. For the ROF model the Euler Lagrange equation was already given as

$$-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|_\epsilon} \right) + \frac{1}{\lambda} (u - f) = 0 \; . \tag{3.22}$$

When we apply the discretisation to Equation 3.22 we get

$$-\,\texttt{div} \left(\frac{(\nabla u)_{i,j}}{|(\nabla u)_{i,j}|_\epsilon} \right)_{i,j} + \frac{1}{\lambda} (u_{i,j} - f_{i,j}) = 0 \; . \tag{3.23}$$

By inserting the definition of the gradient and divergence operator as defined in Equations 3.18 and 3.20 we obtain

$$\begin{aligned} - &\quad \delta_x^- \frac{\delta_x^+ u_{i,j}}{\max\left(\sqrt{\left(\delta_x^+ u_{i,j}\right)^2 + \left(\delta_y^+ u_{i,j}\right)^2}, \epsilon \right)} \\ - &\quad \delta_y^- \frac{\delta_y^+ u_{i,j}}{\max\left(\sqrt{\left(\delta_x^+ u_{i,j}\right)^2 + \left(\delta_y^+ u_{i,j}\right)^2}, \epsilon \right)} \\ + &\quad \frac{1}{\lambda} (u_{i,j} - f_{i,j}) = 0 \; . \end{aligned} \tag{3.24}$$

As already discussed in Section 2.4.2 the fixed point algorithm of Vogel and Oman carries out a linearisation of Equation 3.22. Therefore the non-linear terms in the denominator of the first two parts in Equation 3.24 are taken from the previous step. With the definition of forward and backward differences in Equations 3.19 and 3.21 we obtain the linearised Euler Lagrange equation as

$$\begin{aligned} - &\quad C_1^n \left(u_{i+1,j}^{n+1} - u_{i,j}^{n+1} \right) + C_2^n \left(u_{i,j}^{n+1} - u_{i-1,j}^{n+1} \right) \\ - &\quad C_3^n \left(u_{i,j+1}^{n+1} - u_{i,j}^{n+1} \right) + C_4^n \left(u_{i,j}^{n+1} - u_{i,j-1}^{n+1} \right) + \frac{1}{\lambda} \left(u_{i,j}^{n+1} - f_{i,j} \right) = 0 \; , \end{aligned} \tag{3.25}$$

with the non-linear terms obtained from the previous iteration given as

$$\begin{aligned}
C_1^n &= \frac{1}{\max\left(\sqrt{\left(\delta_x^+ u_{i,j}^n\right)^2 + \left(\delta_y^+ u_{i,j}^n\right)^2}, \epsilon\right)} , \\
C_2^n &= \frac{1}{\max\left(\sqrt{\left(\delta_x^+ u_{i-1,j}^n\right)^2 + \left(\delta_y^+ u_{i-1,j}^n\right)^2}, \epsilon\right)} , \\
C_3^n &= \frac{1}{\max\left(\sqrt{\left(\delta_x^+ u_{i,j}^n\right)^2 + \left(\delta_y^+ u_{i,j}^n\right)^2}, \epsilon\right)} , \\
C_4^n &= \frac{1}{\max\left(\sqrt{\left(\delta_x^+ u_{i,j-1}^n\right)^2 + \left(\delta_y^+ u_{i,j-1}^n\right)^2}, \epsilon\right)} . \qquad (3.26)
\end{aligned}$$

We have now derived a large system of linear equations. One can apply the Jacobi algorithm to efficiently solve this algorithm for $u_{i,j}^n$. Thus we can write our implemented update scheme as

$$u_{i,j}^{n+1,k+1} = \frac{C_1^n u_{i+1,j}^{n+1,k} + C_2^n u_{i-1,j}^{n+1,k} + C_3^n u_{i,j+1}^{n+1,k} + C_4^n u_{i,j-1}^{n+1,k} + \frac{1}{\lambda} f_{i,j}}{C_1^n + C_2^n + C_3^n + C_4^n + \frac{1}{\lambda}} , \qquad (3.27)$$

where k denotes the current iteration number of the Jacobi algorithm. Practise showed that it is sufficient to compute only a few Jacobi iterations for the entire algorithm to converge.

The fixed point algorithm can also be applied to the TV-L^1 model. Here the Euler Lagrange equation is given as

$$-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|_\epsilon}\right) + \lambda \frac{(u-f)}{|u-f|_\delta} = 0 . \qquad (3.28)$$

The differences to the ROF model in Equation 3.22 are marginal. Only an additional weighting of the data fidelity term has to be considered. One can account for this by linearising this term to. Thus the linearised and discretised Euler Lagrange equation of the TV-L^1 model is given as

$$\begin{aligned}
&- C_1^n \left(u_{i+1,j}^{n+1} - u_{i,j}^{n+1}\right) + C_2^n \left(u_{i,j}^{n+1} - u_{i-1,j}^{n+1}\right) \\
&- C_3^n \left(u_{i,j+1}^{n+1} - u_{i,j}^{n+1}\right) + C_4^n \left(u_{i,j}^{n+1} - u_{i,j-1}^{n+1}\right) + \lambda D^n \left(u_{i,j}^{n+1} - f_{i,j}\right) = 0 , \qquad (3.29)
\end{aligned}$$

with the non-linear weighting terms $C_1^n, ..., C_4^n$ given in Equation 3.26, and the newly introduced weighting term for the TV-L^1 data fidelity term is given as

$$D^n = \frac{1}{\max\left(\left|u_{i,j}^n - f_{i,j}\right|, \delta\right)} . \tag{3.30}$$

As the non-linear weighting factors rely solely on forward differences, our algorithm shows a slight anisotropic behaviour. This is a result of our discretisation technique.

3.2.3 Chambolle's Algorithm

We will now take a look at Chambolle's algorithm to solve the dual formulation of the ROF model. In Section 2.4.3.1 we already derived the update scheme for the dual variable $\boldsymbol{p}$ as

$$\boldsymbol{p}^{n+1} = \frac{\boldsymbol{p}^n + \tau\nabla\left(\nabla \cdot \boldsymbol{p}^n + f/\lambda\right)}{1 + \tau\left|\nabla\left(\nabla \cdot \boldsymbol{p}^n + f/\lambda\right)\right|} . \tag{3.31}$$

We can directly apply the discretisation techniques as done above. We obtain the following discretised version of Chambolle's algorithm:

$$\begin{aligned} p_{i,j}^{1,n+1} &= \frac{p_{i,j}^{1,n} + \tau\delta_x^+\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda}\right)}{1 + \tau L_{i,j}^n} \\ p_{i,j}^{2,n+1} &= \frac{p_{i,j}^{2,n} + \tau\delta_y^+\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda}\right)}{1 + \tau L_{i,j}^n} , \end{aligned} \tag{3.32}$$

with

$$L_{i,j} = \sqrt{\left(\delta_x^+\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda}\right)\right)^2 + \left(\delta_y^+\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda}\right)\right)^2} . \tag{3.33}$$

The variable u can be reconstructed at any time as

$$u_{i,j} = f_{i,j} + \lambda\left(\delta_x^- p_{i,j}^1 + \delta_y^- p_{i,j}^2\right) . \tag{3.34}$$

We can further resolve the term

$$\begin{aligned} \delta_x^+\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda}\right) &= p_{i+1,j}^{1,n} - 2p_{i,j}^{1,n} + p_{i-1,j}^{1,n} \\ &+ p_{i+1,j}^{2,n} - p_{i,j}^{2,n} - p_{i+1,j-1}^{2,n} + p_{i,j-1}^{2,n} \\ &+ \frac{f_{i+1,j}}{\lambda} - \frac{f_{i,j}}{\lambda} , \end{aligned} \tag{3.35}$$

and

$$\begin{aligned}
\delta_y^+ \left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} + \frac{f_{i,j}}{\lambda} \right) &= p_{i,j+1}^{1,n} - p_{i,j}^{1,n} - p_{i-1,j+1}^{1,n} + p_{i-1,j}^{1,n} \\
&+ p_{i,j+1}^{2,n} - 2p_{i,j}^{2,n} + p_{i,j-1}^{2,n} \\
&+ \frac{f_{i,j+1}}{\lambda} - \frac{f_{i,j}}{\lambda} \ . \qquad (3.36)
\end{aligned}$$

For implementation this two terms are not necessarily needed.

The extension to the TV-L^1 model is trivial. The thresholding scheme from Equation 2.64 simply becomes

$$v_{i,j}^{n+1} = \begin{cases} u_{i,j}^{n+1} - \lambda\theta & \text{if } u_{i,j}^{n+1} - f_{i,j} > \lambda\theta \\ u_{i,j}^{n+1} + \lambda\theta & \text{if } u_{i,j}^{n+1} - f_{i,j} < -\lambda\theta \\ f_{i,j} & \text{if } \left| u_{i,j}^{n+1} - f_{i,j} \right| \leq \lambda\theta \end{cases} \qquad (3.37)$$

To solve the TV-L^1 model using the Chambolle algorithm we have to iterate the following steps:

- First a few iterations of Equation 3.32 are computed.
- Then u is reconstructed using Equation 3.34.
- Finally the thresholding scheme of Equation 3.37 is applied.

3.2.4 Projected Gradient Descend

The derivation of the projected gradient descend algorithm is straightforward, as in Section 2.4.3.2 we already derived an iterative scheme for the ROF model as follows

$$\begin{aligned}
\tilde{\boldsymbol{p}}^{n+1} &= \boldsymbol{p}^n + (\tau/\lambda) \left[\nabla \left(f + \lambda \nabla \cdot \boldsymbol{p}^n \right) \right] \\
\boldsymbol{p}^{n+1} &= \frac{\tilde{\boldsymbol{p}}^{n+1}}{\max \left\{ 1, \left| \tilde{\boldsymbol{p}}^{n+1} \right| \right\}} \ .
\end{aligned} \qquad (3.38)$$

For the discretised version of the algorithm we first apply the gradient descent of the dual variable $\boldsymbol{p}$

$$\begin{aligned}
\tilde{p}_{i,j}^{1,n+1} &= p_{i,j}^{1,n} + \frac{\tau}{\lambda} \delta_x^+ \left(f_{i,j} + \lambda \left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} \right) \right) \\
\tilde{p}_{i,j}^{2,n+1} &= p_{i,j}^{2,n} + \frac{\tau}{\lambda} \delta_y^+ \left(f_{i,j} + \lambda \left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n} \right) \right) \ . \qquad (3.39)
\end{aligned}$$

In the second step we re-project the dual variable $\boldsymbol{p}$ onto the unit disk

$$\begin{aligned} p_{i,j}^{1,n+1} &= \frac{\tilde{p}_{i,j}^{1,n+1}}{\max\left(1, \sqrt{\left(p_{i,j}^{1,n}\right)^2 + \left(p_{i,j}^{2,n}\right)^2}\right)} \\ p_{i,j}^{2,n+1} &= \frac{\tilde{p}_{i,j}^{2,n+1}}{\max\left(1, \sqrt{\left(p_{i,j}^{1,n}\right)^2 + \left(p_{i,j}^{2,n}\right)^2}\right)} . \end{aligned} \tag{3.40}$$

To solve the TV-L^1 model using the projected gradient descend algorithm we have to iterate the following steps:

- First compute a few iterations of Equation 3.39 and 3.40.
- Then u is reconstructed using Equation 3.34.
- Finally the thresholding scheme of Equation 3.37 is applied.

3.2.5 Segmentation

With the knowledge from the above derivations, it is very simple to discretise the proposed segmentation algorithm. The algorithm is described in detail in Section 3.1.2.

First we have to solve for u using the projected gradient descend algorithm in Equation 3.12. In the discretised version we first apply the gradient descent

$$\begin{aligned} \tilde{p}_{i,j}^{1,n+1} &= p_{i,j}^{1,n} + \frac{\tau}{\theta}\delta_x^+ \left(f_{i,j} + \theta\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n}\right)\right) \\ \tilde{p}_{i,j}^{2,n+1} &= p_{i,j}^{2,n} + \frac{\tau}{\theta}\delta_y^+ \left(f_{i,j} + \theta\left(\delta_x^- p_{i,j}^{1,n} + \delta_y^- p_{i,j}^{2,n}\right)\right) . \end{aligned} \tag{3.41}$$

After that, the dual variable $\boldsymbol{p}$ is reprojected using the edge information g

$$\begin{aligned} p_{i,j}^{1,n+1} &= \frac{\tilde{p}_{i,j}^{1,n+1}}{\max\left(1, \frac{\sqrt{\left(p_{i,j}^{1,n}\right)^2 + \left(p_{i,j}^{2,n}\right)^2}}{g_{i,j}}\right)} \\ p_{i,j}^{2,n+1} &= \frac{\tilde{p}_{i,j}^{2,n+1}}{\max\left(1, \frac{\sqrt{\left(p_{i,j}^{1,n}\right)^2 + \left(p_{i,j}^{2,n}\right)^2}}{g_{i,j}}\right)} . \end{aligned} \tag{3.42}$$

Then u is reconstructed as

$$u_{i,j}^{n+1} = v_{i,j}^{n} + \theta \left(\delta_x^- p_{i,j}^{1,n+1} + \delta_y^- p_{i,j}^{2,n+1} \right) \ . \tag{3.43}$$

After this we update v with the following threshold scheme

$$v_{i,j}^{n+1} = \begin{cases} u_{i,j}^{n+1} - \lambda_{i,j}\theta & \text{if} \ \ u_{i,j}^{n+1} - f_{i,j} > \lambda_{i,j}\theta \\ u_{i,j}^{n+1} + \lambda_{i,j}\theta & \text{if} \ \ u_{i,j}^{n+1} - f_{i,j} < -\lambda_{i,j}\theta \\ f_{i,j} & \text{if} \ \ \left| u_{i,j}^{n+1} - f_{i,j} \right| \leq \lambda_{i,j}\theta \ . \end{cases} \tag{3.44}$$

To solve the proposed segmentation model to iterate the following steps:

- First 3 to 5 iterations of Equation 3.41 and 3.42 have to be computed.
- Then u is reconstructed using Equation 3.43.
- Finally the thresholding scheme of Equation 3.44 is applied.

When one looks at the algorithm above, it can be seen that the extension to three dimensions is trivial.

Chapter 4

Implementation

Contents

4.1 Hardware

We already mentioned that the algorithms discussed in the previous section, are perfectly suited for implementation on parallel hardware. In the following we show that a parallel approach is seminal, as the current development in computer hardware leads towards massive parallelisation. Our implementation was done on the graphics device. Therefore we will first review the history of graphics hardware, and then discuss the advantages of state-of-the-art hardware architecture.

4.1.1 History

Graphics hardware has to deal with a huge amount of data to render realistic 3D scenes. For graphical applications a high throughput has to be accomplished to render scenes interactively at high frame rates. 3D scenes always require the same workflow that has to be performed. As different stages have to be computed one after the other, the streaming processor model is perfectly suited for this task. There is the need to perform one and the same arithmetic operations on multiple data e.g. pixels. In turn, the massive parallelisation potential of rendering tasks led graphics hardware towards more and more parallelisation on the hardware side. As there is no need for complex control instructions, more transistors can be used on the data path. Memory access usually follows a strict pattern, therefore memory is optimised to deliver high throughput but has bad latency times when data is required that diverges from the standard access pattern. With deep pipelines the latency time can be hidden efficiently. On the other hand, a standard CPU is designed for general purpose applications. Thus it is optimised to do efficient branch control for serial programs. It offers more complex instruction sets than a GPU and has much higher control requirements as different threads are competing for a single CPU. Recently CPUs are beginning to get parallelised by incorporating more than one core on a single chip. As there is usually no regular pattern of memory access, memory is optimised for latency.

In the early years, graphic hardware offered a fixed rendering pipeline that could not be altered. In order to compute more complex illumination models programmable shader units were introduced at the vertex and pixel levels. They offered the possibility to apply a small program to each pixel or vertex. A standard programmable graphics pipeline is shown in Figure 4.1. At the output of the graphics pipeline the rendered result can be stored into a texture that can be used in the next pass. For the first time it was possible to "abuse" the GPU for general purpose computations.

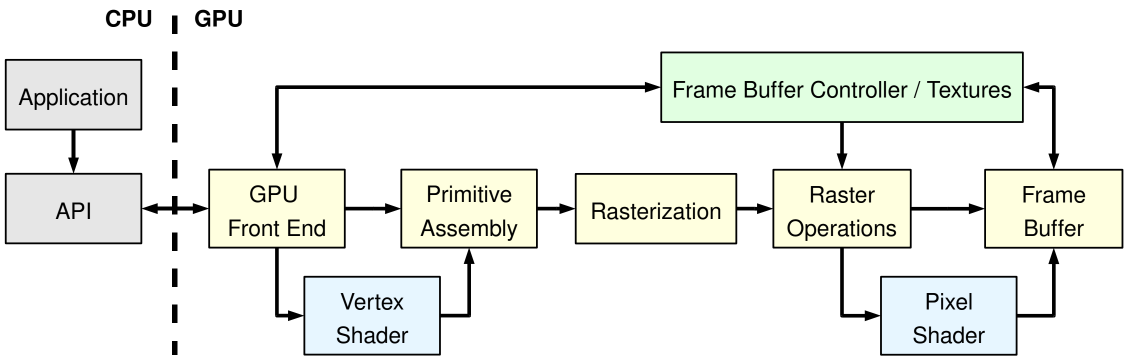


Figure 4.1: The traditional programmable graphics pipeline.

To write programs for the programmable shading units, shading languages had to be used together with OpenGL or DirectX. Various languages emerged to ease the programming of the shaders. The most common and platform independent language is CG [38] that was developed by NVidia. As a very similar language HLSL [1] was developed by Microsoft for DirectX. GLSL [75] is a more recent development by 3DLabs. All this languages still require knowledge of the graphics hardware and the usage of OpenGL or DirectX. Languages were developed that allow general purpose computations on different parallel hardware, with the aim to be able to concentrate solely on the algorithm, and not on the implementation on the graphics device. The most common ones are Accelerator [85], Brook [18], Sh [60], CTM [2], Shallows [79], CUDA [69] and RapidMind [73]. Though there are great differences between these languages they have the common aim to abstract the programming from the underlying architecture while still making use of its advantages.

Recently also variational methods were implemented on graphics hardware [72]. They showed the potential of fast GPU implementations by making some of them applicable for real time problems. In contrast to languages that were originally intended for computer graphics as in [72], we used the general purpose framework CUDA on the newest generation of graphics hardware, that will be described in the following.

4.1.2 Used Hardware

The latest generation of GPUs combines all shader units in the so-called unified shader architecture. In Figure 4.2 the different programmable units of the rendering pipeline are replaced by a single processing unit that is used for different purposes. The new

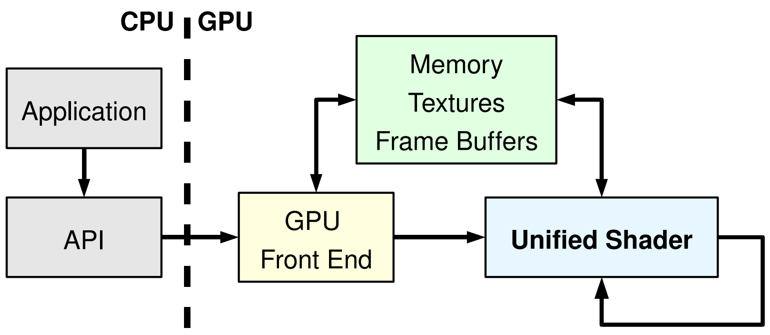


Figure 4.2: The unified graphics pipeline introduced along with the GeForce 8-series.

architecture is designed to allow general purpose programs to be executed. This new architecture has been introduced by NVidia the first time with the GeForce 8-series [68]. Along with the new architecture they provided the Compute Unified Device Architecture (CUDA) language [69], that we will discuss in Section 4.2.

On the GeForce 8-series the processing units of the GPU are arranged into groups of so-called multiprocessors. One multiprocessor, can execute several independent threads having access to the same shared memory. The number of threads that are executed physically in parallel is referred to as the warp size. A half-warp is the first or second half of a warp and equals the number of processors in a single multiprocessor. The streaming architecture ensures that the processors are used to full capacity.

State-of-the-art graphics hardware offers the impressive performance of up to 576 GFlops and a memory bandwidth of up to 103.7GB/s. When one takes a close look to the development of graphics hardware, one can see that the computational performance increases more rapidly than the memory bandwidth (see Figure 4.3). On the GeForce 8800 already 24 floating point operations can be performed while a single operand is fetched from the global GPU memory. This means that algorithms with high arithmetic intensity are well suited to be computed on the GPU.

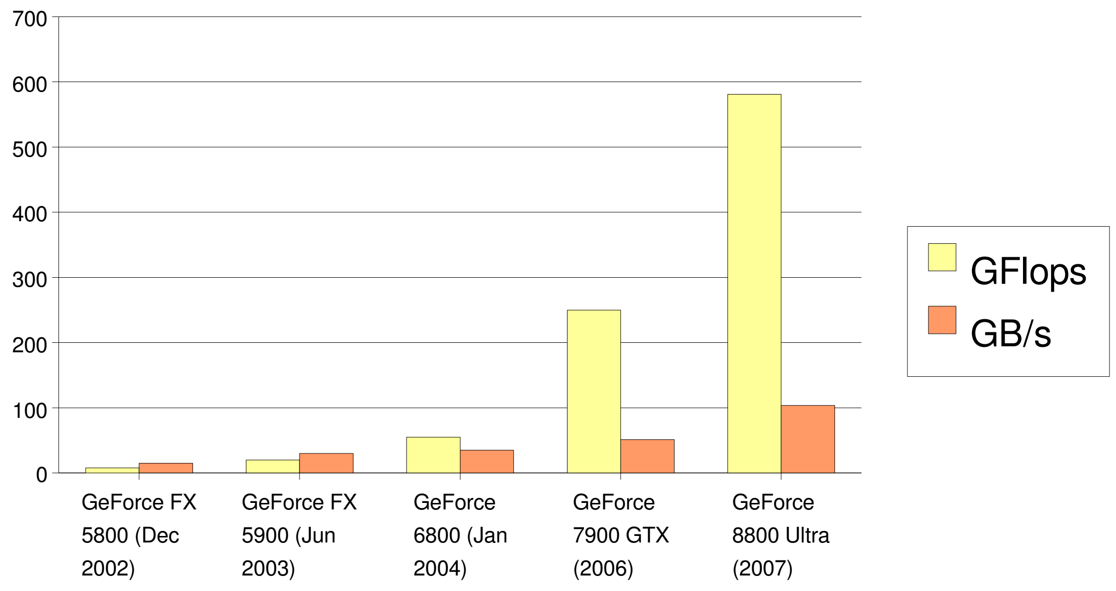


Figure 4.3: The computational power of graphic cards over time.

In Figure 4.4 different kinds of memory are illustrated as provided to the user for programming. The host (CPU) can only exchange data with the global memory, the constant memory and the texture memory. Data transfers between the host and the graphics device are very slow (Peak bandwidth is 4GB/s with PCI-express x16). Texture memory and constant memory are read-only areas. While the texture memory provides fast access and bilinear interpolation for 1D/2D spatial access patterns with high latency, the small constant memory is very fast for data that is acquired by all threads of a warp at once. Shared memory is arranged in banks that can provide data parallel. When correctly accessed shared memory is as fast as reading from the registers.

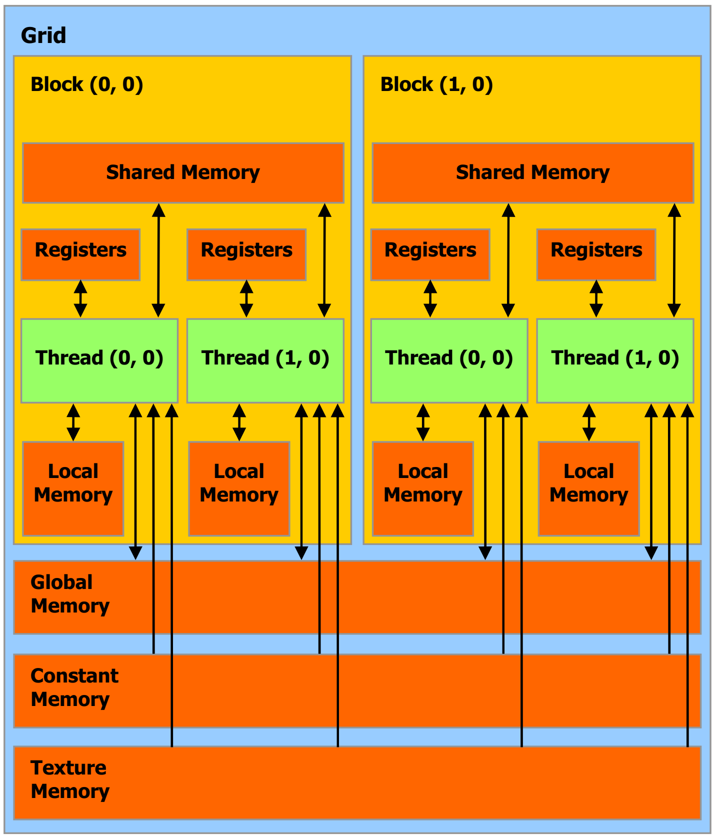


Figure 4.4: Memory as seen by the user on the NVidia GeForce 8 architecture. The figure was taken from [69].

4.2 CUDA

With the introduction of the GeForce 8-series, NVidia also introduced the CUDA (Compute Unified Device Architecture) framework. CUDA provides a standard C language interface for programming on the GPU, that handles scheduling and execution. It can handle a massive number of parallel threads that are scheduled to the processor. We will now review basic properties of the CUDA framework, and discuss some general performance issues.

4.2.1 The Framework

Applications that require the same computations for a lot of different data, meaning that there is a big number of identical threads, are perfectly suited to be implemented using CUDA. Threads can be arranged in blocks of up to three dimensions. These thread blocks are executed on a single multiprocessor, and can thus communicate through the shared memory. Blocks are further organised in a grid of blocks that is up to two dimensional. This grid corresponds to the execution of a single CUDA program that is also referred to as kernel. This principle is depicted in Figure 4.5. The kernel executes the same program to different data, with the threads being identified by their location in the grid and blocks. There is no way of determining when each block is executed, thus communication between blocks is not possible. If communication is required serialisation can be achieved by consecutive kernel calls and communication can be done through global memory. Multiple kernels can be executed at the same time on a single device.

The CUDA framework handles thread execution, memory allocation and memory transfers. The execution of algorithms on the GPU has only marginal overhead that has to be executed on the CPU. The CPU can therefore be used for other tasks. For image processing tasks, the grid - block structure is perfectly suited, as threads can be easily assigned to pixels/voxels. Although already a simple implementation can gain big speedups compared to the implementation on a CPU, the architecture has to be taken into account to gain optimal performance. In the following we will discuss some general performance issues.

4.2.2 General Performance Issues

We found that a few general strategies have to be followed to gain optimal performance of the implementation. The most important issue is to maximise parallelism. In the case of high parallelism, the high latency of memory access can be overlapped with computations. The number of blocks used by an algorithm should be at least the number of multiprocessors available, thus no processors run idle. In order to keep the multiprocessors busy all the time, multiple blocks running concurrently on a single multiprocessor are desirable. Although more threads per block will help to hide memory latency, shared memory space and registers get a limiting factor. Best performance depends on the certain type of algorithm and has to be determined by experiments. What one can say in general is that the number of threads per block should be a multiple of the warp size, as otherwise computational power is wasted.

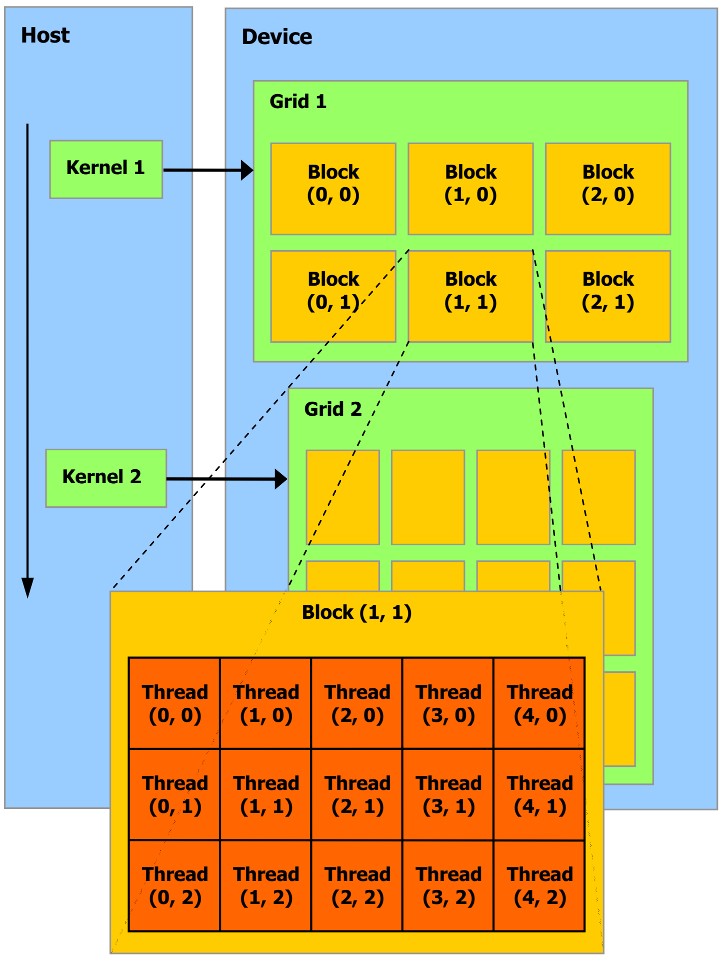


Figure 4.5: Organisation of threads into blocks that are executed on a single multiprocessor. The blocks are organised in a grid, and all execute the same program. The figure was taken from [69].

Another important issue is memory usage (see also Section 4.1). Especially transfers between CPU and GPU are very costly. They should be avoided if possible, even if that means that kernels with low parallelism have to be computed on the GPU. Also on the device side, exchanging of data can be very costly. Each thread should be as independent from other threads as possible, especially if they need to communicate through global memory. Threads that have to communicate should do this using the shared memory, by keeping them on a single multiprocessor. As this is not always possible kernels have to be spilt up and communicate using the global memory. It is in some cases more efficient to recalculate data than to move it around. This effect will increase in the future, as the development of graphics hardware shows in Figure 4.3. A typical CUDA kernel may first load data from global memory into shared memory. Then the data is processed using only the fast shared memory. Possibly several iterations can be performed before the results are written back to global memory.

As global memory is not cached, one large data transfer is always faster than a lot of small ones. Moreover the right access pattern is crucial for an effective bandwidth. CUDA can load 128-bit words from global memory in a single instruction. Acquisition of memory has to be organised in a way that the simultaneously requested data can be coalesced into as few as possible memory accesses. Coalescing is possible if consecutive threads acquire consecutive memory forming a contiguous block in memory. This is especially important for a half warp. But also the starting address of the acquired memory should be aligned to the half warp size times the elements size. If the memory is structured in higher dimensional grids, padding of the data is required to fulfil the alignment requirements. Experiments showed high performance gains when aligning memory for an optimal access pattern.

Furthermore the shared memory requires some consideration. On a multiprocessor unit many processors acquire memory at the same time. Therefore shared memory is divided into banks that each can service one address per cycle. When multiple kernels access a bank at the same time this results in bank conflicts. When a bank conflict occurs accesses have to be serialised. Thus bank conflicts have a minor impact compared to wrong global memory access patterns, additional performance can be gained by minimising them.

4.3 Implementation Details

We will now discuss our implementation on the graphics hardware. First we will present our strategies for optimal usage of memory. As already mentioned in the previous sections, great performance gains can be achieved when utilising shared memory. Furthermore we will discuss the user interface and its consequences for the segmentation task in depth.

4.3.1 Utilising Shared Memory

We already discussed guidelines for optimal performance in Section 4.2.2. For the total variation implementations only the neighbouring pixels are needed to update the current pixel. Therefore we chose the approach to load a patch into shared memory and perform several iterations on it before writing it back to global memory. We will refer to iterations done on shared memory as internal iterations. As for the high access speed to the shared memory, the more internal iterations the more overall iterations can be computed in a given time. Of course this speed gain will saturate when the number of internal iterations gets very high. Figure 4.6 shows the number of overall iterations per second depending on the number of internal iterations. One can also see that the primal versions can compute much more iterations per second, as computations are more simple. Another limitation is given by the fact that at the borders of the thread blocks no synchronisation can be performed. While the interior of the block converges towards a solution, it cannot influence the pixels in the next block. The interaction between single thread blocks happens only after the fixed number of internal iterations. This attenuates the diffusion process leading to an overall higher convergence time, though more overall iterations are carried out. If the number of internal iterations gets too high, errors may be introduced, and the algorithm will not converge towards the correct solution any more. In the worst case oscillations may occur at the borders of thread blocks. Of course one could exchange the border of the thread block through the global memory. But as the sequence of execution for the thread blocks is not specified, this method is not deterministic. Furthermore only a few neighbouring blocks are executed at the same time. Experiments showed that, when the number of internal iterations gets too high, though a higher number of internal iterations could be computed, the convergence time increased.

The optimal number of internal iterations was determined by experiments. Therefore the current solution was compared to a reference solution, as described in Section 5.2. Figure 4.7 shows such a comparison of the convergence time in dependence of internal iterations for the implemented denoising algorithms. One can see that while the processing time of the algorithms needed for convergence at first decreases, it starts increasing again if the number of internal iterations gets to high. For the primal versions this happens very slowly, and one could also carry out much more internal iterations without any significant slowdown. The dual versions on the other hand will introduce errors earlier, and will then soon start to oscillate at thread block borders. The dual TV-L^1 algorithm showed a different behaviour than all of the other algorithms. It has to be mentioned that we need at least three internal iterations to get sufficient results. Otherwise the optimisation parameter θ has to be decreased, making the algorithm slower. For some values of λ the processing time increased almost linearly, and for others it decreased a little bit before increasing significantly. Thus it is save to chose the number of internal iterations as low as possible. The optimal number of internal iterations slightly varied for different values of λ.

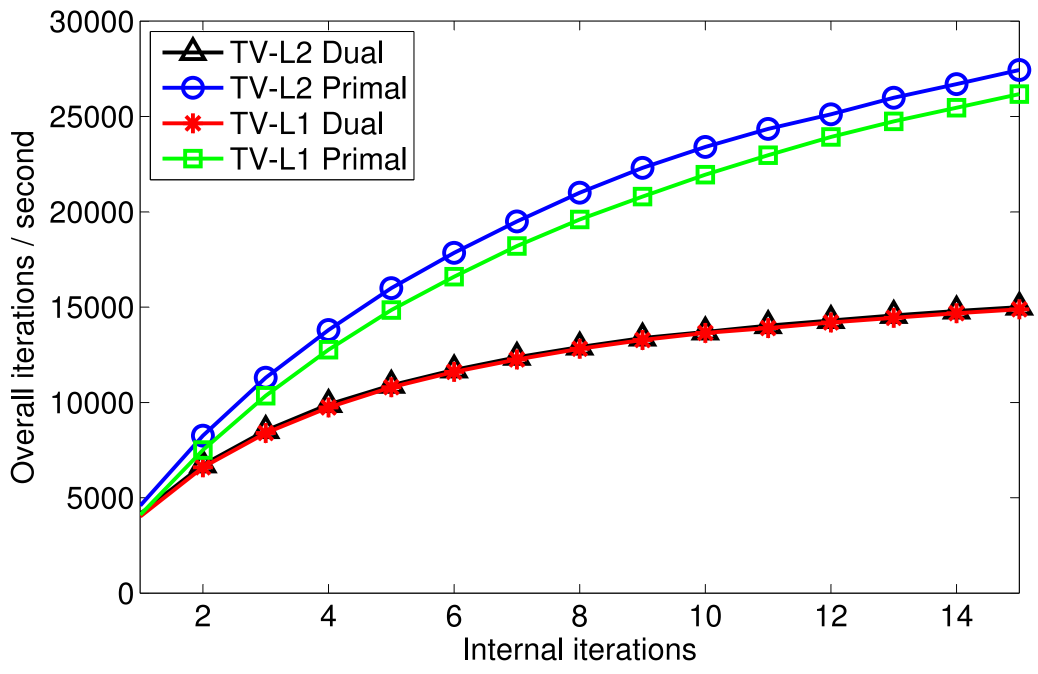
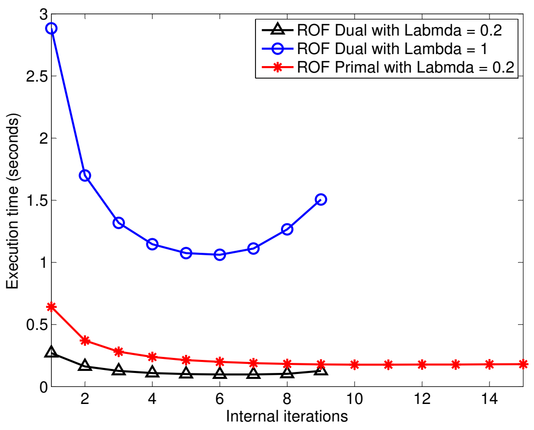
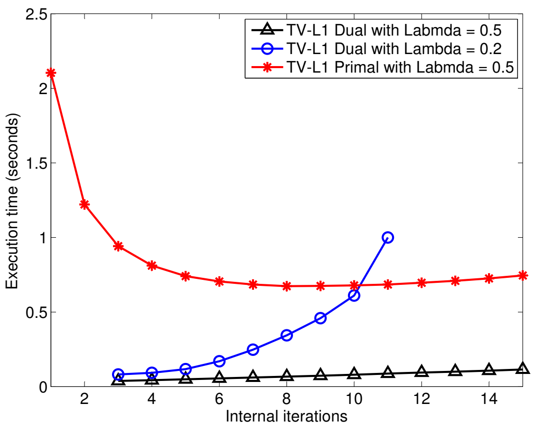


Figure 4.6: Number of overall iterations per second for different numbers of internal iterations.



(a)



(b)

Figure 4.7: Comparison of convergence speed for different numbers of internal iterations for (a) ROF model and (b) TV-L^1 model.

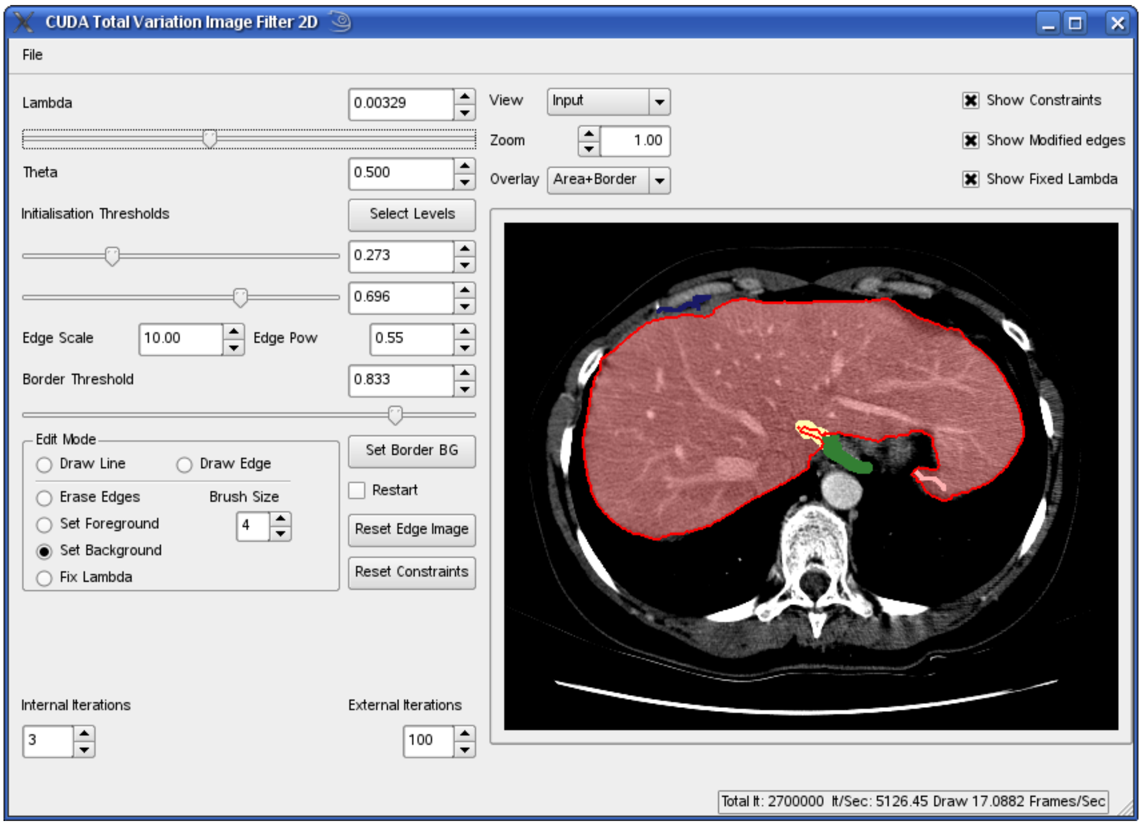
We fixed them to deliver a good compromise, especially for the commonly used values of λ, as 5 for the dual ROF algorithm and 3 for the dual TV-L^1 and 10 for the primal ones. As the segmentation algorithm is based on the dual TV-L^1 implementation, the number of internal iterations for the segmentation algorithm was chosen as 3 too.

4.3.2 Graphical User Interface

A graphical user interface was implemented using QT, to be able to efficiently work with the algorithms. The user has at any time the full control over all parameters of the algorithm. Changes of parameters immediately take effect allowing a real time experience with all algorithms. For image denoising the interface is straightforward, and we therefore focus only on the segmentation task. There are several ways to obtain a segmentation. If the desired object is very large and has a homogeneous gray value range different from the background, one could do segmentation using weak constraints. By a single click on the image the gray level range is selected automatically. By varying λ the desired shape smoothing can be obtained. If the optimal segmentation cannot be achieved by this way, additional local constraints can be incorporated. For 2D problems, the interface provides the following functionality:

- *Foreground / Background*: This hard constraints force the pixels to be of a certain class. In the case of the pure geodesic energy, at least one foreground and one background seedpoint has to be set.
- *Erase* Edges: By modifying the edge image with the erase tool, strong edges that are near the desired segmentation border can be deleted.
- *Draw* Edges (freehand or lines): If certain edges are too weak or missing one can draw additional edges to the image.
- *Fix* λ: In unconstrainted areas a global value for λ is set. If one wants to preserve very fine details in some parts of the segmentation, while smoothing it in other parts, this function allows to locally fix λ.

In Figure 4.8 a screenshot of the segmentation process is shown. The pale red areas are foreground constraints and the dark blue areas are background constraints. They were used here to modify the segmentation result obtained through weak constraints. The green areas indicate that the edges were modified. In this case they were deleted. In the yellow area λ was fixed at a higher value to preserve finer structures of the border.



Lambda 0.00329

Theta 0.500

Initialisation Thresholds 0.273 0.696

Edge Scale 10.00 Edge Pow 0.55

Border Threshold 0.833

Edit Mode: ○ Draw Line ○ Draw Edge ○ Erase Edges ○ Set Foreground [✓ ● Set Background] ○ Fix Lambda

Brush Size 4

☐ Restart

Internal Iterations 3

External Iterations 100

View Input

Zoom 1.00

Overlay Area+Border

☒ Show Constraints

☒ Show Modified edges

☒ Show Fixed Lambda

Total It: 2700000 It/Sec: 5126.45 Draw 17.0882 Frames/Sec

Figure 4.8: A screenshot of the 2D segmentation application during the segmentation process.

Another way one can approach the segmentation is to completely discard the grey value information by choosing $\lambda = 0$. We then calculate the pure geodesic energy. In this case the user has to set a foreground and a background seed. If the segmentation is still not perfect one can still add more hard constraints or modify the edge image. It showed that the segmentation approach minimising the pure geodesic energy is much more efficient, as any possible region can be selected not depending on gray value constitution.

For 3D segmentation problems the user can view only a single slice at once. Datasets can be viewed in axial, sagittal and coronal view. As in 3D, the weak constraints approach seemed not that useful, we only implemented the *Foreground* / *Background* and the *Erase* brush. The user can draw to a specified number of planes at once.

An efficient workflow is guaranteed by different overlay modes of the segmentation, e.g. area highlighting, or borders. Of course the drawing indication can be hidden for the user to see the original image all the time. For a better understanding of the algorithm any variable can be visualised. For the segmentation process it might sometimes be helpful to view the edge image (or weight function), as the final segmentation highly depends on the quality of the edges. The weight function was chosen as in Equation 3.2. In Figure 4.9 the edge image is shown for different values of β and $\alpha = 1$. Pixels with a high gray value are located at edges and correspond to a low cost. It showed that $\beta = 0.55$, as already proposed by [48], worked sufficiently for all of our segmentations.

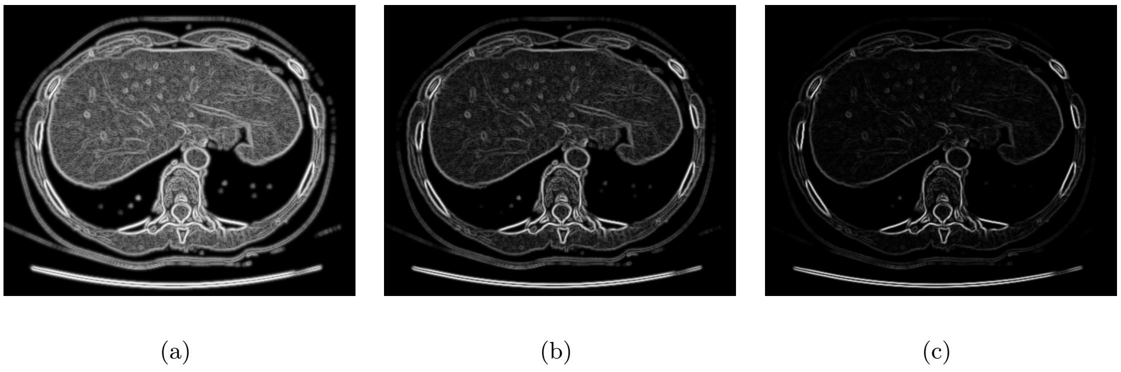


Figure 4.9: The edge image with different values of β and $\alpha = 1$: (a) $\beta = 0.25$, (b) $\beta = 0.55$ and (c) $\beta = 1$.

In medical datasets the grey value range is usually very high. Commonly there are far more gray values than the 256 gray values that can be displayed on a standard display. Therefore the user can select a range that is scaled between the lowest and highest displayable value. Every gray value above or below is cropped. Relevant edge information can sometimes be very small compared to other edges in the image. Therefore the interface provides the possibility to recalculate the edges based on the current view. Experiments showed that adapting the view to relevant structures before calculating edge information highly improves the segmentation process, or sometimes even makes it possible, and thus easily adapts the segmentation process to the users intention.

Chapter 5

Results

Contents

5.1 Some Preliminaries

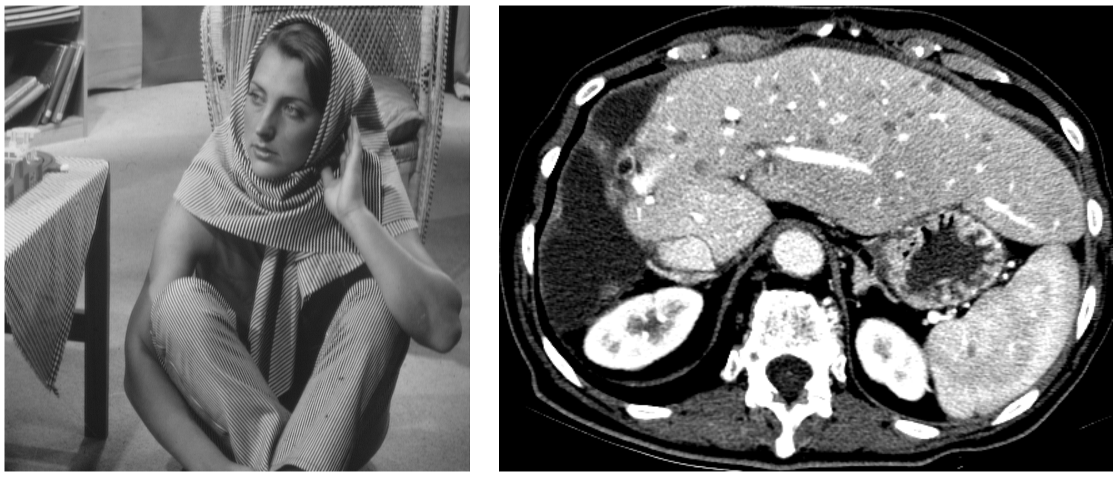
In the following, experimental results will be presented that show the capability of the proposed segmentation framework and the implementations of the denoising algorithms on the GPU. As the proposed segmentation method is interactive, exact time measurements are not meaningful. This problem is not existent in image denoising tasks. Therefore exact speed evaluations will be presented only for the denoising algorithms. In case of segmentation, examples of the effectiveness of the algorithm are presented and only approximate times needed to obtain the results are given.

All experiments were done on a single PC with an Intel dual core processor with a clock speed of 2.66 GHz, and 4 GB RAM. The system was equipped with a standard NVidia GeForce 8800 GTX as the graphics processor. We used a recent version (1.1) of CUDA. We note that in the past already performance gains were achieved by updates of CUDA.

5.2 Speed Tests on Image Denoising

Exact speed evaluations of the implementations were only done on the denoising algorithms. This was done in comparison to the work of Goldfarb and Yin in [40] that is to our knowledge one of the fastest implementations of TV methods in two and three dimensions. For the same reason we chose the *Barbara* image, as depicted in Figure 5.1(a), in 2D, and the 3D dataset that we will refer to as *Brain*. The size of the original image is 512×512. Additionally we show results on a medical image of the size 454×344 that we will refer to as *Liver2D*, and is depicted in Figure 5.1(b).

5.2.1 Denoising in 2D



(a) (b)

Figure 5.1: (a) The original 512×512 *Barbara* image. (b) The 454×344 *Liver2D* image

To find a good measurement on how long the algorithms should be iterated, several possibilities were taken into account. One way, as done by [23] is to look for changes of the optimisation variable during the iteration process. If they are small enough one can assume that the optimal solution is reached. Of course this can only be applied to the dual formulation. Another possibility is to calculate the Euler Lagrange equation, that equals zero if the optimal solution is found. Due to numerical inaccuracies this solution will probably never be reached exactly. Furthermore it is not very useful to wait for an exact solution of the floating point calculations if the input image is only a 8 bit image.

Therefore a threshold can be introduced as a stopping criterion. But it is not evident how this threshold should be chosen. A third method is to compare the current result to a reference solution. The algorithm is iterated until the difference to the ground truth gets below a certain stopping criterion. We chose to do the evaluation by comparison with a ground truth, as one can define a reasonable stopping criterion. The error was calculated as the root mean squared error. To obtain the ground truth, the algorithm was iterated a very long time to make sure to find the true solution. The stopping criterion for 8 bit images was chosen as 0.001, that equals the quantisation of the image gray levels for a 10 bit image that is scaled between $[0, 1]$. We examined different other stopping criterions, as can be seen for the ROF model in Table 5.1. One can see that we could still gain a

Algorithm	ROF - Primal				ROF - Dual			
Stopping Criterion	11 bit	10 bit	9 bit	8 bit	11 bit	10 bit	9 bit	8 bit
Iterations	5370	4130	2940	1830	2050	1065	490	210
Iterations/sec	23400				10900			
Time $[ms]$	229	176	126	78.2	168	97.7	45	19.3

Table 5.1: Different stopping criterions for the ROF algorithms for *Barbara* with $\lambda = 0.2$.

very high speedup, up to the factor 5, by weakening the measurement criterion. In Figure 5.2 the dual ROF model is applied to the *Barbara* image with fixed numbers of iterations according to Table 5.1. The images are crops of the complete image showing the most critical area of this image, that is here the tablecloth on the left hand side. Visually one can only notice differences with the 8bit stopping criterion as depicted in Figure 5.2(b). But this errors might still be insignificant for a lot of applications in computer vision. Especially when we see the speed evaluations in context with the segmentation problem, a very low accuracy is needed for our purposes.

The TV-L^1 model, that is directly related to the segmentation task, showed a very similar behaviour on the variation of the stopping criterions, as can be seen in Table 5.2.

Algorithm	TV-L^1 Primal				TV-L^1 - Dual			
Stopping Criterion	11 bit	10 bit	9 bit	8 bit	11 bit	10 bit	9 bit	8 bit
Iterations	23670	14720	8800	4820	453	315	216	138
Iterations/sec	21950				8390			
Time [ms]	1078	671	401	220	54	37.5	25.7	16.4

Table 5.2: Different stopping criterions for the TV-L^1 algorithms on *Barbara* with $\lambda = 0.5$.

To determine the speed of the 2D algorithms against the number of pixels, the Barbara image was cropped taking only the centre part to 256×256 containing a quarter of pixels

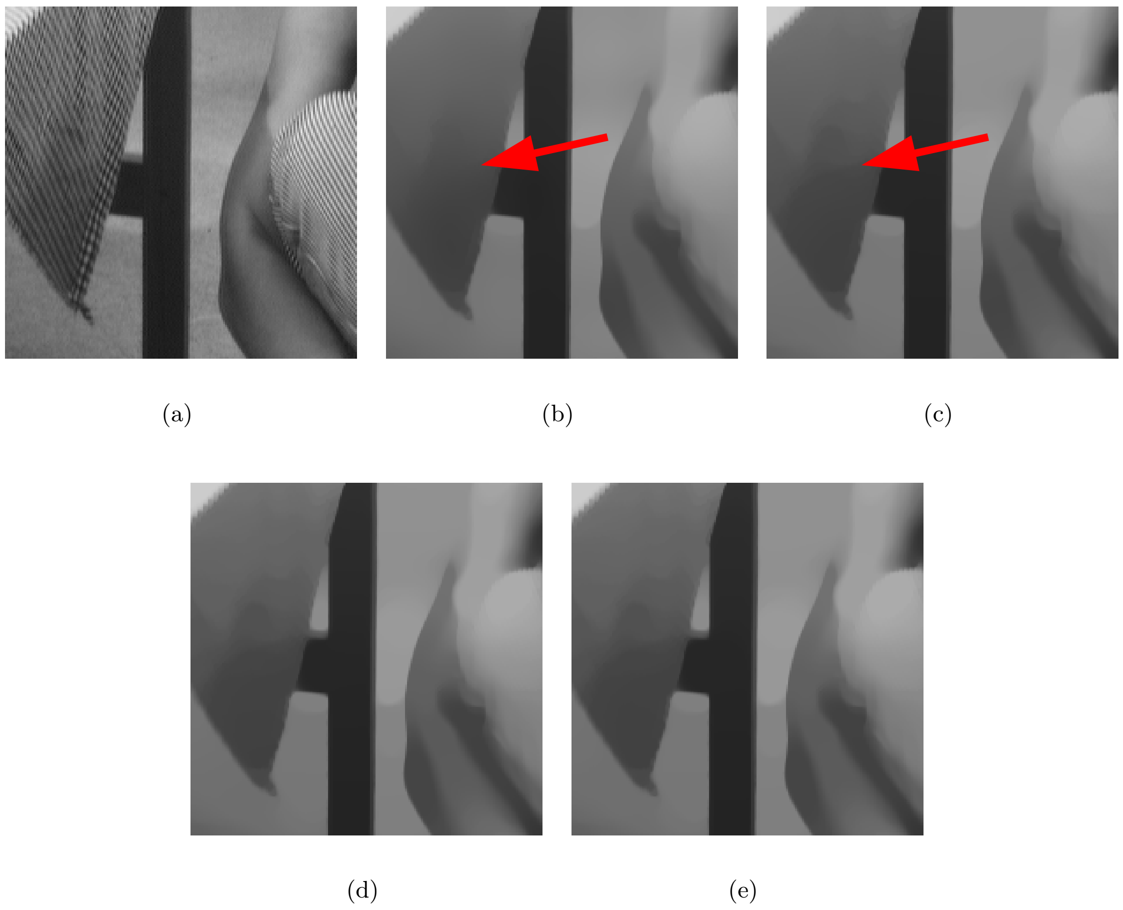


Figure 5.2: The dual ROF model applied to *Barbara* for different stopping criterions: (a) crop of original image, (b) 8 bit: 210 iterations, (c) 9 bit: 490 iterations, (d) 10 bit: 1065 iterations and (e) 11 bit: 2050 iterations.

and extended it to 1024×1024 by replicating it four times. A comparison with the original image and the *Liver2D* image, with different values for λ can be found in Table 5.3 for the dual ROF model, and in Table 5.4 for the dual TV-L^1 model. It shows that the

Image	*Barbara*						*Liver2D*	
Size	256×256		512×512		1024×1024		454×344	
λ	0.5	0.2	0.5	0.2	0.5	0.2	0.5	0.2
Iterations	5685	1090	4280	1070	4950	1210	6000	1785
Iterations/sec	42400		10900		2780		17650	
Time $[ms]$	134	25.7	393	98.1	1781	435	340	101

Table 5.3: Comparison of different image sizes with the dual ROF algorithm applied on *Barbara* and *Liver2D*

Image	*Barbara*						*Liver2D*	
Size	256×256		512×512		1024×1024		454×344	
λ	0.5	0.2	0.5	0.2	0.5	0.2	0.5	0.2
Iterations	294	675	315	681	324	579	1398	1656
Iterations/sec	31730		8390		2106		13525	
Time $[ms]$	9.3	21.3	37.5	81.2	154	275	103	122

Table 5.4: Comparison of different image sizes with the dual TV-L^1 algorithm applied on *Barbara* and *Liver2D*

algorithm run time is almost linear in the total number of pixels. One can also notice that convergence speed does not only depend on the number of pixels, but also on the content. Although the *Liver2D* image is smaller than the *Barbara* image, more iterations are needed to find the solution, leading to a higher overall convergence time. This is due to the algorithm depending on the image gradient. In homogeneous regions the algorithm converges much slower than in case of highly textured areas. This effect shows more when using the TV-L^1 model. A direct comparison between TV-L^1 and ROF model cannot be performed, as they deliver quite different results.

Convergence speed also strongly depends on the value of λ. The more smoothing is applied, the longer it takes to compute the solution. In Table 5.5 the ROF model was evaluated for different values of λ. Table 5.6 shows the results for the TV-L^1 model. One can notice that the primal versions of the algorithms take significantly more time than the dual ones. An interesting point is the increase of convergence time for high values of λ with the dual TV-L^1 algorithm. Although only very small image structures are removed, the time to find the optimal solution starts to increase. We yet do not know the reason for this effect.

As already mentioned earlier, we chose to use the projected gradient descend updates over the Chambolle updates as one achieves better convergence times. Table 5.7 gives a justification by showing the convergence times for different values of λ when using Chambolle updates. The resulting convergence times can be directly compared to the results in Table 5.5 and 5.6. One can note that the effect is more significant when using the ROF model.

In order to be comparable to the work of Goldfarb and Yin in [40], that defined their ROF model as

$$\min_u \left\{ \int_\Omega |\nabla u| \, d\Omega + \lambda' \int_\Omega (u - f)^2 \, d\Omega \right\} , \tag{5.1}$$

Algorithm	ROF - Primal				ROF - Dual			
λ	0.05	0.1	0.2	1.0	0.05	0.1	0.2	1.0
Iterations	570	1640	4130	31500	100	310	1070	11150
Iterations/sec	23400				10900			
Time $[ms]$	24.3	70.1	176	1346	9.2	28.4	98.1	1023

Table 5.5: Comparison of runtime for the ROF model with different values for λ applied on *Barbara*

Algorithm	TV-L^1 - Primal				TV-L^1 - Dual			
λ	2.0	1.0	0.5	0.1	2.0	1.0	0.5	0.1
Iterations	8530	7050	14710	84600	3522	483	315	1242
Iterations/sec	21950				8390			
Time $[ms]$	389	321	670	3854	420	57.6	37.5	148

Table 5.6: Comparison of runtime for the TV-L^1 model with different values for λ applied on *Barbara*

Algorithm	ROF - Dual				TV-L^1 - Dual			
λ	0.05	0.1	0.2	1.0	2.0	1.0	0.5	0.1
Iterations	175	630	2260	23200	3501	453	336	2319
Iterations/sec	10890				8250			
Time $[ms]$	16	57.8	670	2130	424	54.9	40.7	281

Table 5.7: Comparison of runtime for the TV-L^1 model using Chambolle updates with different values for λ applied on *Barbara*.

and they worked on a gray scale range between 0 and 255, we have to convert λ as

$$\lambda = \frac{1}{2\lambda' 255} \ . \tag{5.2}$$

As the TV-L^1 model is contrast invariant, and they defined it the same way we did, the value of λ is directly comparable. The code from [40] can be downloaded from `http://www.caam.rice.edu/~wy1/ParaMaxFlow/`. With the original code, we achieved on our PC runtimes that are twice as fast as they claimed in their paper. For the 2D images we always used the 16-neighbourhood, as for a 4-neighbourhood the metrication error is too high as one can see in Figure 5.3 by the block artifacts of the output.

In Table 5.8 our results are compared to the work of Goldfarb and Yin [40], Pock et al. [72] and a Matlab implementation. Note that some times are given in milliseconds and others in seconds. We tested the CG implementation of Pock et al. on our current hardware. Compared to the results from [72] slight speedups could be gained. The Matlab

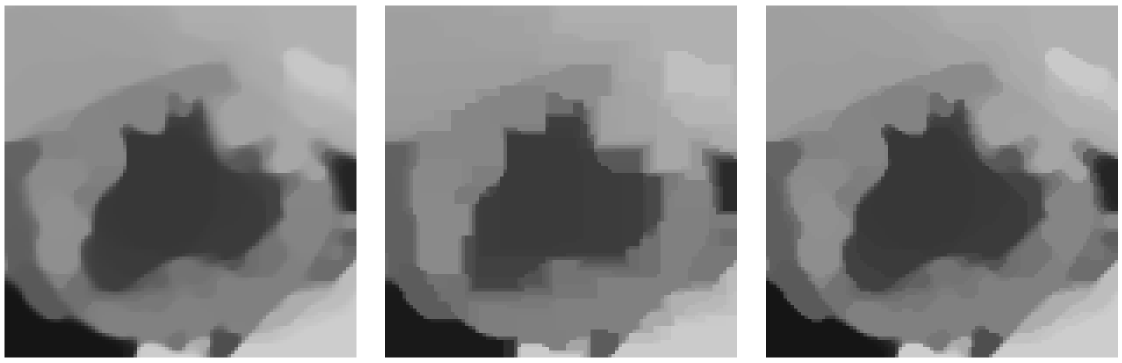


Figure 5.3: A crop of the dual ROF model applied to *Liver2D* using (a) our implementation, (b) the implementation of Goldfarb and Yin [40] with 4-neighbourhood and (c) 16-neighbourhood.

implementation is approximately 1000 times slower than our implementation when using the 10 bit stopping criterion for both implementations as described above. Compared to the implementation of Goldfarb and Yin our approach is $10 - 100$ times faster with the ROF model, and approximately 80 times faster for the TV-L^1 model. When comparing to the CG implementation from [72] we are slightly more than 10 times faster with the ROF model. With the TV-L^1 model even higher speedups of up to 100 can be gained. Our results with the 8 bit stopping criterion that have more significance to the segmentation task, gain an additional speedup. Form this data we can assume that our implementation using CUDA is currently the fastest TV denoising implementation.

Algorithm	ROF				TV-L^1			
λ	0.05	0.1	0.2	1.0	2.0	1.0	0.5	0.1
Our implementation (10 bit) [ms]	9.2	28.4	98.1	1023	420	57.6	37.5	148
Our implementation (8 bit) [ms]	1.8	6	19.2	293	94	20.4	16.4	57.5
Pock et al. [72] [s]	0.11	0.37	1.36	12.6	1.65	2.46	5.6	26
Goldfarb and Yin [40] [s]	2.08	2.89	3.9	9.12	0.3	0.74	1.1	3.19
Matlab [s]	9.75	33.8	124	1154	542	808	1846	n.a.

Table 5.8: Comparison of runtime with other algorithms on *Barbara*.

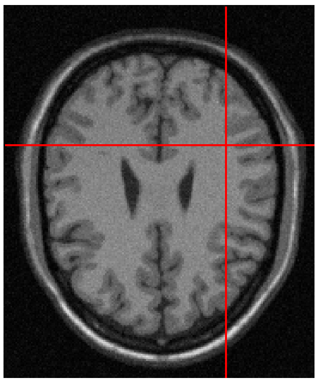
5.2.2 Denoising in 3D

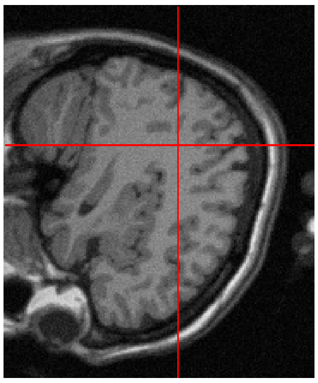
For the 3D denoising evaluation we used the *Brain* and the *Liver3D* dataset. Some slices of the original datasets are shown in Figure 5.4, where the *Liver3D* dataset shows only values between 0 and 255 HU. The *Brain* dataset is a synthetical MRI T1 image with 5% Gaussian noise that can be downloaded from BrainWeb [14]. Its size is $182 \times 217 \times 181$ and thus has approximately 7.1 million voxels that are of the size $1 \times 1 \times 1$ mm. The gray values are in the range from 0 to 32767. Actually the image is a 8 bit image with rescaled values. Therefore we rescaled them back to 0 to 255. Additionally tests were done with the *Liver3D* dataset having a size of $512 \times 512 \times 60$ that has approximately 15.7 million voxels of the size $0.55 \times 0.55 \times 2$ mm. To be better comparable between the two datasets, we ignored the original pixel size, and treated the voxels as if they were of the size $1 \times 1 \times 1$ mm. The *Liver3D* dataset has gray values from -1024 to 1401 HU, that were scaled between $[0, 1]$.

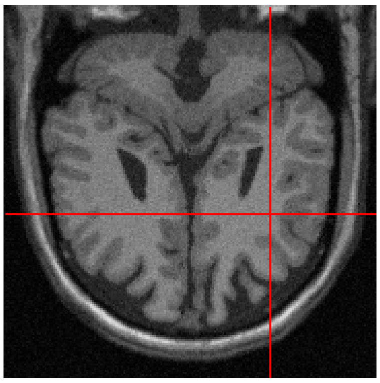
As the *Brain* dataset has the same gray value range than the 2D test images, we left the stopping criterion at 10bit. The *Liver3D* dataset has more gray values, and would therefore make a bigger error. Figure 5.5 shows the results of applying the TV-L^1 model with $\lambda = 0.2$ to the *Liver3D* dataset with different stopping criterions. The slice shown is the same as depicted in Figure 5.4(d). One can notice that with 10bit stopping criterion small differences to the other images are visible. When we view this in context of the segmentation task, even a smaller stopping criterion may be possible. We chose the stopping criterion for the *Liver3D* dataset with 11bit.

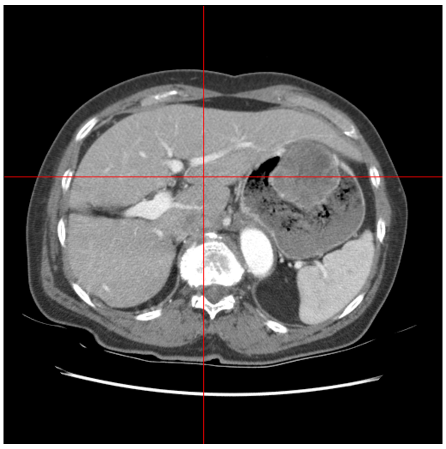
The ROF model was tested for different values of λ. The results for the *Brain* dataset can be found in Table 5.9. The same experiments were also done for the *Liver3D* dataset, and results can be seen in Table 5.10. For the 3D datasets we do not present results with Chambolle iterations, as the effects are the same as in two dimensions. We use the projected gradient descend updates for all experiments. Once more we can notice that the dual algorithms outperform the primal ones. Note that the speed gain, that can be achieved, also depends on the data. One can notice that the gain is up to 10 times higher for the *Brain* dataset compared to the *Liver3D* dataset.

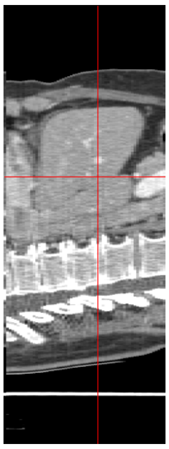
The TV-L^1 model was evaluated in Table 5.11 for the *Brain* dataset and in Table 5.12 for the *Brain* dataset. For the TV-L^1 model it showed that the algorithm does not depend on the data as strong as the ROF model. Speedups of up to the factor 10 can be gained by using the dual approach. Once more the dependence of convergence speed on λ is noticeable. In the case of the dual TV-L^1 experiments with the *Brain* dataset,


(a)


(b)


(c)


(d)


(e)

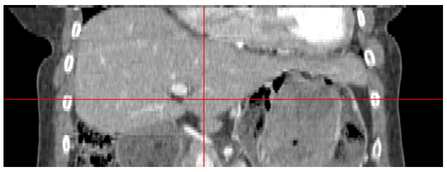

(f)

Figure 5.4: The *Brain* dataset at (a) plane 100 in axial view, (b) plane 128 in sagittal view and (c) plane 80 in coronal view. The *Liver3D* dataset, showing only gray values between -200 and 300 HU, at (d) plane 35 in axial view, (e) plane 232 in sagittal view and (f) plane 199 in coronal view.

Algorithm	ROF - Primal			ROF - Dual		
λ	0.01	0.05	0.1	0.01	0.05	0.1
Iterations	1200	4000	18400	20	220	470
Iterations/sec	307			131		
Time $[s]$	3.9	13	60	0.15	1.68	3.59

Table 5.9: Comparison of runtime for the ROF model with different values for λ applied to *Brain*.

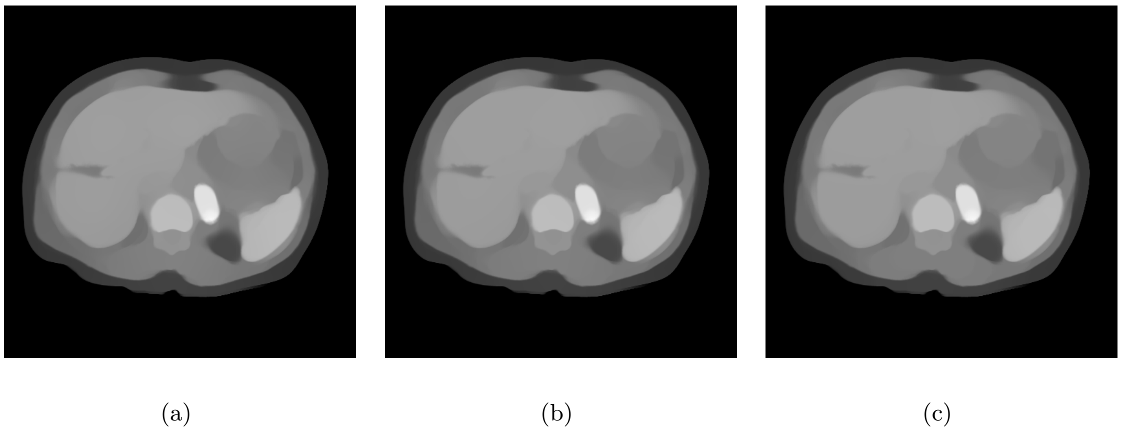


Figure 5.5: The TV-L^1 model applied to *Liver3D* with $\lambda = 0.2$ for different stopping criterions: (a) 10 bit: 984 iterations, (b) 11 bit: 2322 iterations and (c) 12 bit: 6330 iterations.

convergence speed increases continuously with decreasing λ. On all other experiments in Tables 5.11 and 5.12 convergence speed starts to increase for very high values of λ.

Algorithm	ROF - Primal			ROF - Dual		
λ	0.01	0.05	0.1	0.01	0.05	0.1
Iterations	200	4760	9640	50	1010	3890
Iterations/sec	137			57.5		
Time [s]	1.46	34.7	70.4	0.87	17.6	67.6

Table 5.10: Comparison of runtime for the ROF model with different values for λ applied to *Liver3D*.

Algorithm	TV-L^1 - Primal			TV-L^1 - Dual		
λ	1.0	0.5	0.2	1.0	0.5	0.2
Iterations	12900	11290	41500	264	912	1056
Iterations/sec	304			63.5		
Time [s]	42.4	37.1	137	4.1	14.4	16.6

Table 5.11: Comparison of runtime for the TV-L^1 model with different values for λ applied on *Brain*.

Our implementation was compared to the work of Goldfarb and Yin in Table 5.13. Only the *Brain* dataset was used for this comparison, as there was not enough memory to compute *Liver3D* with the maximum flow algorithm. It has also to be mentioned, that for 3D a 6-neighbourhood was the only available option, and therefore the results suffer from

Algorithm	TV-L^1 - Primal			TV-L^1 - Dual		
λ	1.0	0.5	0.2	1.0	0.5	0.2
Iterations	40380	29220	61750	2184	810	1560
Iterations/sec	134			26.5		
Time $[s]$	301	218	460.8	82.4	30.6	58.9

Table 5.12: Comparison of runtime for the TV-L^1 model with different values for λ applied on *Liver3D*.

strong metrication errors. This can be seen by a comparison with our version in Figure 5.6. Especially the TV-L^1 model suffers from a too small neighbourhood. Our implementation using the 10 bit stopping criterion is up to 190 times faster for the ROF model, and up to 8 times faster for the TV-L^1 model. In Figure 5.6 results are compared with the maximum flow algorithm. With the 8 bit stopping criterion one could gain another speedup of up to 4 for the TV-L^1 model, and even higher speedups for the ROF model.

Algorithm	ROF			TV-L^1		
λ	0.01	0.05	0.1	1.0	0.5	0.2
Our implementation 10bit $[s]$	0.15	1.68	3.59	4.1	14.4	16.6
Our implementation 8bit $[s]$	0.08	0.23	0.53	1.8	5.57	6.94
Goldfarb and Yin [40] $[s]$	28.5	106	124	32.3	50.1	142

Table 5.13: Comparison of runtime with other algorithms on *Brain*.

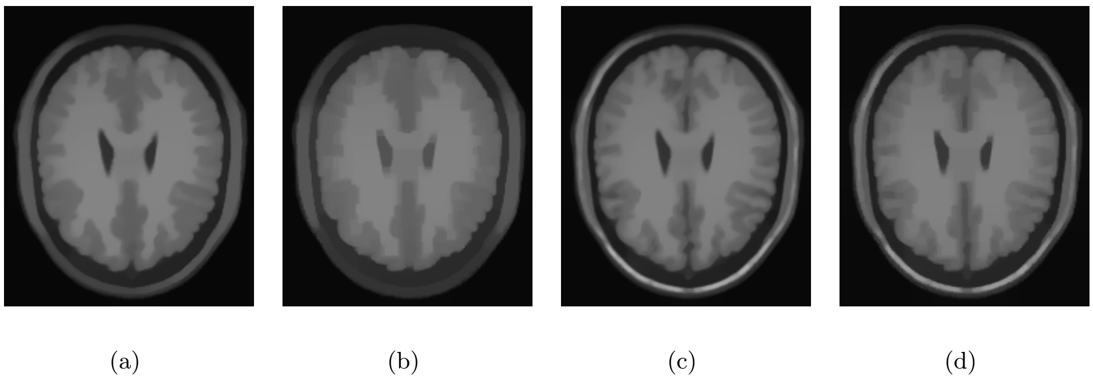


Figure 5.6: Comparison of our algorithm to the maximum flow algorithm of Goldfarb and Yin: TV-L^1 model with $\lambda = 0.5$ (a) our algorithm and (b) maximum flow. ROF model with $\lambda = 1.0$ (c) our algorithm and (d) maximum flow.

5.2.3 Discussion

The results in this section showed that the proposed algorithms for image denoising are very fast. Compared to graph based methods, we do not suffer from any metrication error. Comparable graph based methods using a high neighbourhood connection are up to 200 times slower than our implementation. Further more memory consumption of graph based methods is much higher than with our proposed algorithm. It also showed that the CUDA implementation using current hardware is much more efficient than implementations using one of the older general purpose languages.

The algorithms showed almost linear runtime behaviour on the number of pixels/voxels. When one compares the 2D with the 3D example, we can see from experiments that per voxel more time is needed for the 3D algorithm. As an example we look at the convergence time for the 1024×1024 (1 million pixels) image in Table 5.4 for $\lambda = 0.5$, that is 154 ms. On the other hand the convergence time for the $182 \times 217 \times 181$ (7.1 million voxels) dataset in Table 5.11 is 14.4 ms for $\lambda = 0.5$. The convergence time per pixel is 13 times higher for the 3D algorithm. This is due to the higher number of memory accesses, and more complex computations that have to be performed in 3D. We also noticed that convergence time depends on the data, as the updates are driven by image gradients. Of course the convergence time also depends on the choice of λ. Generally one can say that the larger the image structures that have to be removed, the more time is needed.

As experiments showed, convergence times strongly depend on the choice of the stopping criterion. In context to the segmentation that will be discussed in the next section, the TV-L^1 model with the 8 bit stopping criterion gives the best hint on the possible speed.

5.3 Segmentation Evaluation

In the following examples using the developed segmentation tools are presented. As already mentioned exact speed evaluations were not done as the segmentation is interactive, and depends highly on the user. To show the relation between the dual TV-L^1 algorithm and the segmentation algorithm, the number of iterations, that are both capable to perform, are listed in Table 5.14. One can note that in the 2D case, the dual TV-L^1 algorithm can perform approximately 1.6 times more iterations than the 2D segmentation algorithm. This is partly due to the higher computational costs, but mainly caused by the higher number of memory needed by the algorithm. In the 3D case the loss of speed is much

Image	2D			3D	
Size	256×256	512×512	1024×1024	$182 \times 217 \times 181$	$512 \times 512 \times 60$
TV-L^1	31730	8390	2106	63.5	26.5
Segmentation	19560	5200	1355	56	26

Table 5.14: Comparison of the number of iterations per second between the dual TV-L^1 algorithm and the segmentation algorithms.

lower. This is due to the more efficient memory implementation. As for large 3D datasets memory is getting scarce, variables such as f or the array containing the constraints were merged with other variables. Though more computations have to be done to extract the different variables from the merged arrays, speed can be gained. This also emphasises the need for a sufficient memory management.

5.3.1 Segmentation using Weak Constraints

In Section 4.3.2 different methods on how to approach a segmentation were already mentioned. Here we will first go into detail on how to obtain a segmentation using weak constraints.

In Figure 5.7 an example is given by segmenting a lung in a CT image. With a single click on the image the thresholds are initialised as can be seen in Figure 5.7(a). One can note that in this example almost all relevant tissue is selected as a part of the segmentation. On the other hand a lot of other structures are selected as they have similar gray values. By applying shape denoising on the weak constraints, as achieved by lowering λ, the segmentation as depicted in Figure 5.7(b) can be obtained. All small structures are removed making the result more accurate. But still the strong edges at the ribs attract the border in the upper left part of the image. After deleting this edges the border will snap back to the next strong border. Another problem occurs at the notch almost in the centre of the image. Because of the small value of λ the structure is removed, and the border takes a “shortcut” with a lower cost. By simply fixing λ at a higher value, this problem can be removed. On the right hand side of the image a small lobe is not part of the segmentation as the gray values differ too much for the thresholding approach. By using the foreground brush this tissue can be added to the segmentation. In Figure 5.7(c) the final segmentation is shown along with the added local constraints. This segmentation can be obtained in a few seconds. Figure 5.7(d) shows the variable u. The segmentation was obtained by extracting the level-set $u = 0.5$. Especially when looking at the borders

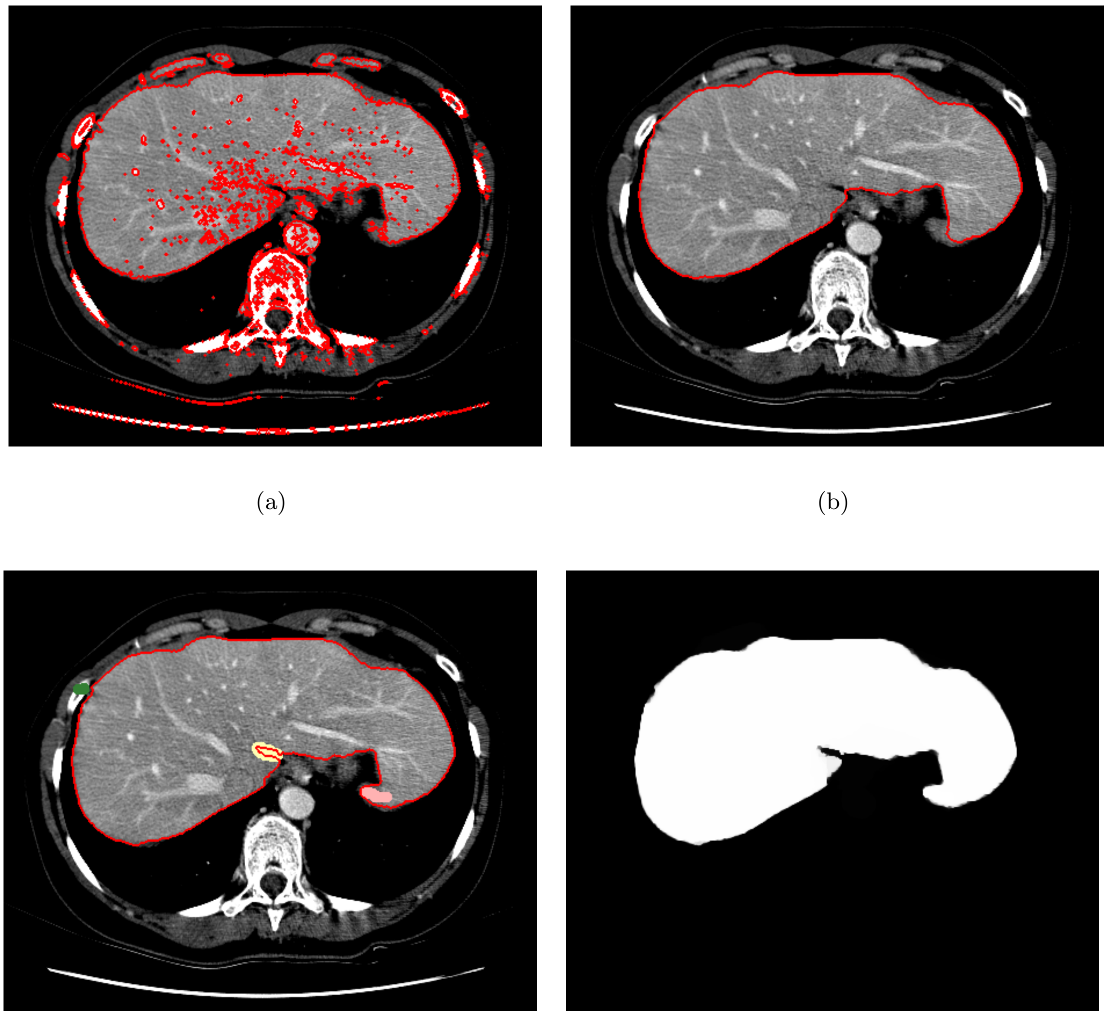


Figure 5.7: Segmentation using weak constraints in 2D for a medical example: (a) threshold segmentation with high value of λ, (b) decreased λ, (c) final segmentation with added constraints and (d) the final segmentation represented by u.

of the segmentation, one can note that u is not a binary image. Other solutions can be obtained by selecting another level-set of u.

In Figure 5.8 a non-medical segmentation is presented. The segmentation process is very similar to the previous example. After adding a few local constraints, the final segmentation is obtained as in Figure 5.8(c) within a few seconds. To illustrate the effects of smoothing (or shape denoising) on the initial seed region, Figure 5.8(d) depicts the final segmentation overlaid to f that was obtained by thresholding.

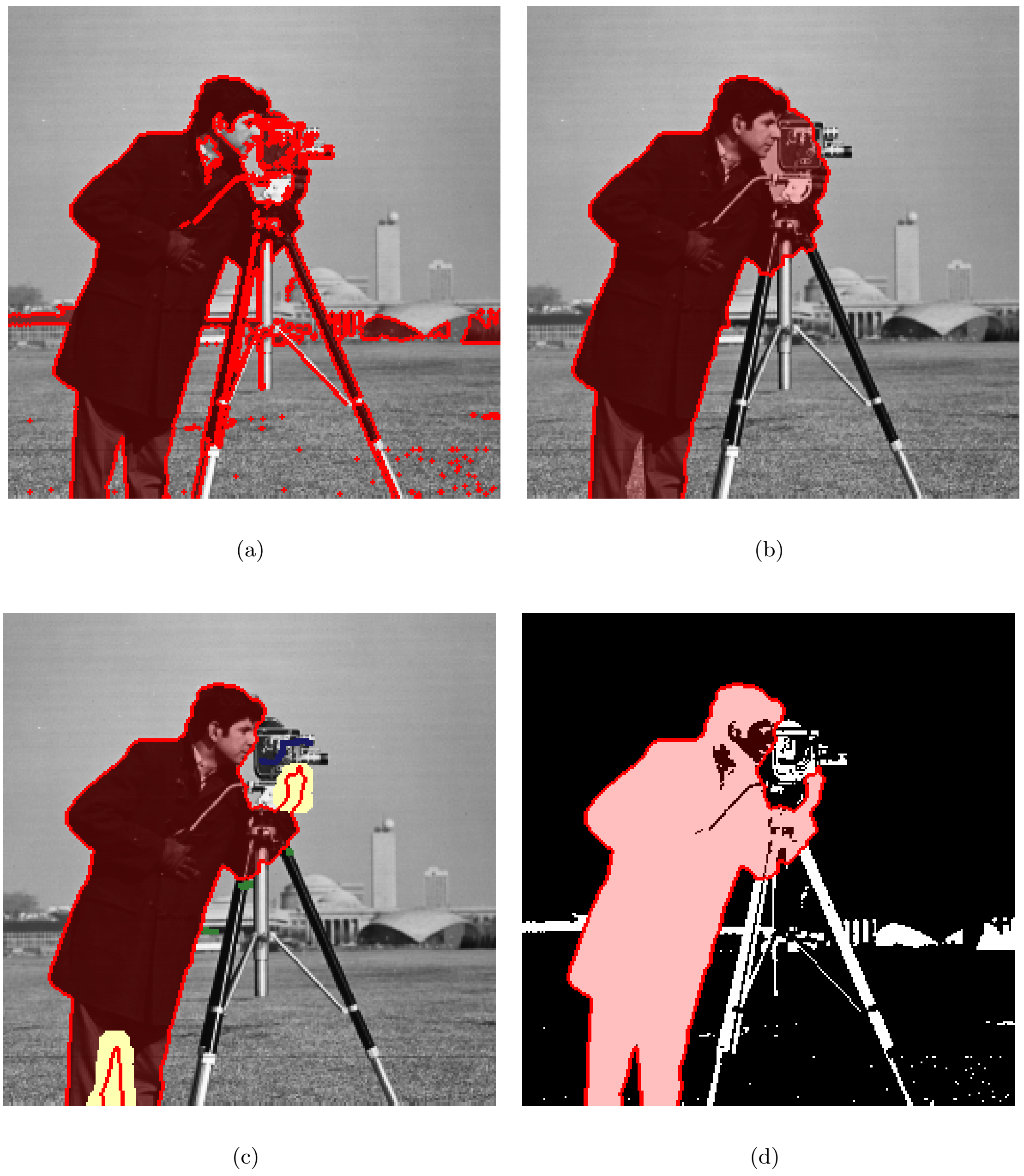


Figure 5.8: Segmentation using weak constraints in 2D for a real world example: (a) threshold segmentation with high value of λ, (b) decreased λ, (c) final segmentation with added constraints and (d) the final segmentation overlaid on f.

Experiments showed that the most useful brushes for the segmentation approach using weak constraints, are the *Background*, the *Edge* erase and the *Fix* λ brush. The possibility to draw edges was only useful in very few cases. Furthermore the *Foreground* brush was not used very often, but can sometimes help to obtain the desired segmentation (see Figure 5.7).

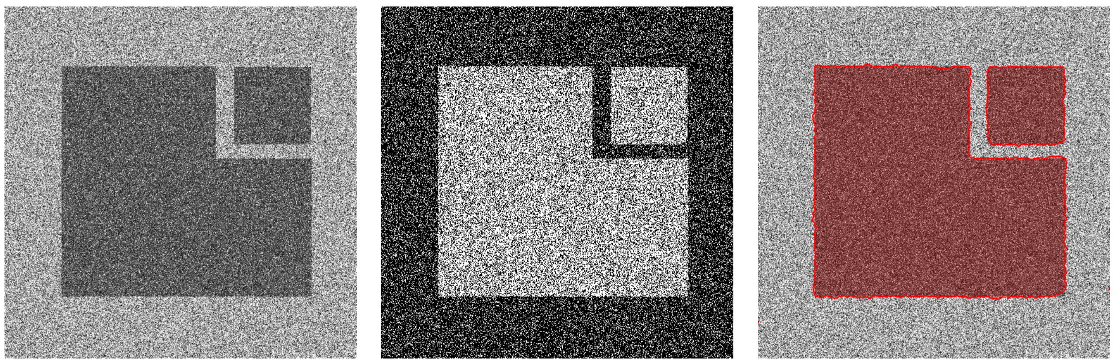

(a) (b) (c)

Figure 5.9: Segmentation using weak constraints for an artificial example: (a) the original image, (b) f obtained by thresholding and (c) the segmented image.

Artificial data was created to test the capabilities of the segmentation approaches. In Figure 5.9(a) two black blocks on a white background are corrupted by heavy noise. There are no significant borders making edge information useless. Nevertheless, when looking at Figure 5.9(b) that visualises f obtained by thresholding, it is obvious that a good segmentation can be easily found by smoothing the weak constraints obtained by thresholding. Figure 5.9(c) shows the final segmentation using the weak constraints approach.

As a final example for segmentation using weak constraints in 2D we will take a view at the example of Figure 1.3 presented in Section 1.4.1. One can see that the thresholding does not lead to a satisfying result. When using solely weak constraints, the segmentation is sure to fail. By incorporating hard constraints, such as additional foreground and background seeds, a segmentation can be obtained as one can see in Figure 5.10. For this example approximately 30 seconds were needed to draw all the constraints and find the final segmentation. Segmentation using weak constraints is not very convenient for this example. In the next section we will show that this image can be segmented more easily and much faster if we rely solely on the pure geodesic energy.

We will present only one 3D example here. Though weak constraints can be used for a lot of medical examples, usually the approach using solely geodesic energy is more efficient. In Figure 5.11 the *Liver3D* dataset already presented in Section 5.2.2 was segmented. For

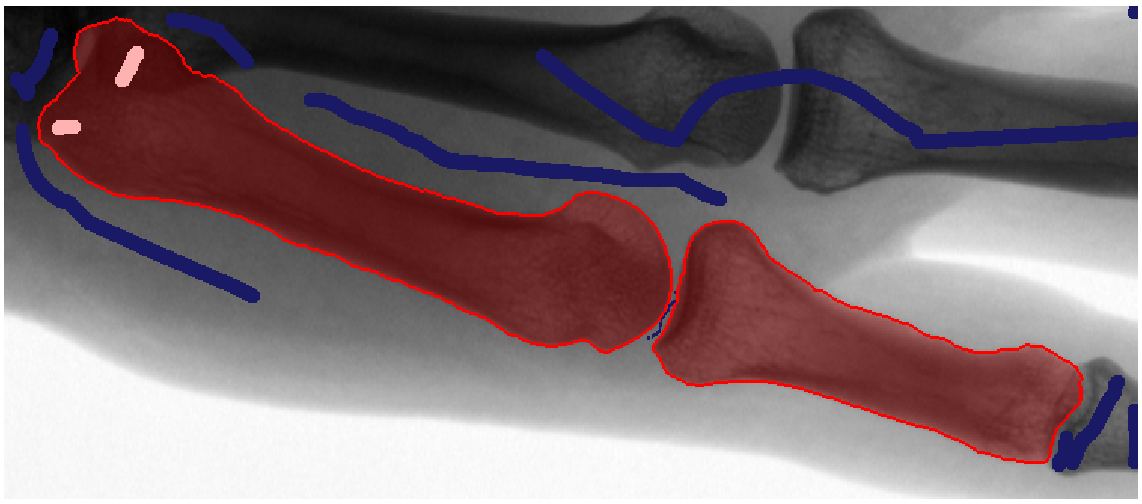


Figure 5.10: Two bones of the *Bone* image obtained using weak constraints and a lot of hard constraints.

some sample slices see Figure 5.4. The liver was segmented using thresholding to obtain the initial seed region. This dataset has the advantage that the liver can be characterised by a quite homogeneous gray value range. Nevertheless a lot of local constraints had to be incorporated, to exclude some parts from the segmentation. Also the edge erase functionality proved well for refining the contour. The segmentation was obtained within 5 minutes.

5.3.2 Segmentation using Pure Geodesic Energy

By choosing $\lambda = 0$ segmentation using solely the geodesic energy is performed. Without constraints no practical segmentation can be obtained this way. At least one foreground and background seed is necessary to get a segmentation. In Figure 5.12 the *Bone* image from Figure 5.10 is segmented using the pure geodesic energy. The border is automatically set as background. With only two small foreground seeds, the final segmentation is obtained. With this approach only a few seconds are needed to get the final segmentation. Thus this method is for this image superior and much more practical than the approach using weak constraints. In Figure 3.2 (that was already depicted in Section 3.1.2) the evolution of the variable u over time is shown. The shown evolution process took approximately one second.

To demonstrate the effectiveness of the various implemented tools, Figure 5.13 shows how an incorrect segmentation using solely geodesic energy can be modified to get the

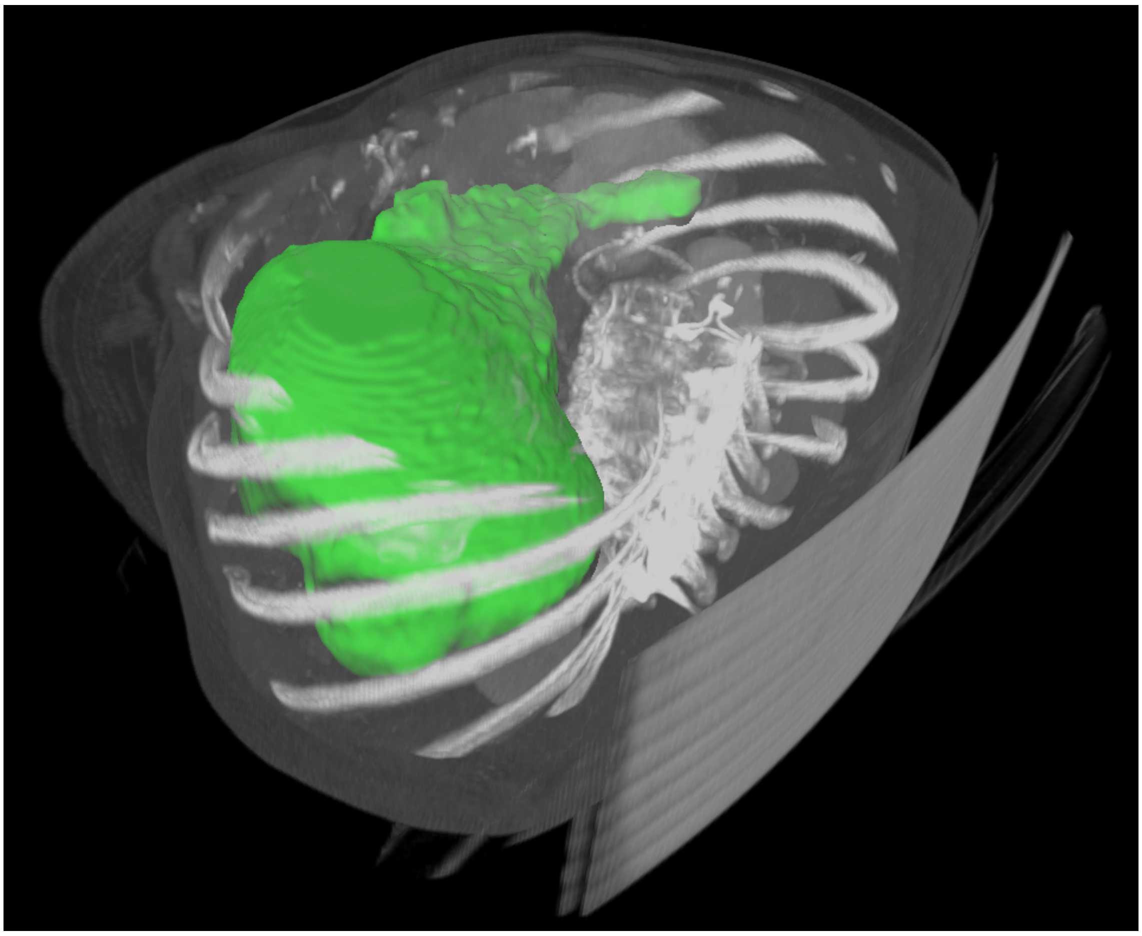


Figure 5.11: The liver was segmented in the *Liver3D* dataset using weak constraints.

desired segmentation by using the edge modification tools. We therefore used a CT image where we want to segment the liver. Figure 5.13(a) shows an example where the selection of foreground seeds does not provide enough information to yield the desired segmentation of the liver. Here a foreground seed was set in the liver, and the border of the image was set to background. By using the erase brush, the edge causing the wrong segmentation is deleted effecting the algorithm to snap to the next stronger edge. Similar, the drawing tool is used to introduce an additional edge that effectively attracts the segmentation border (see Fig. 5.13(c)). Since our algorithm is guaranteed to yield a global optimum, the algorithm immediately converges against the new optimum after drawing onto the image. Fig. 5.13(b) depicts the result of the new global optimum after taking into account the local constraints provided by the user.

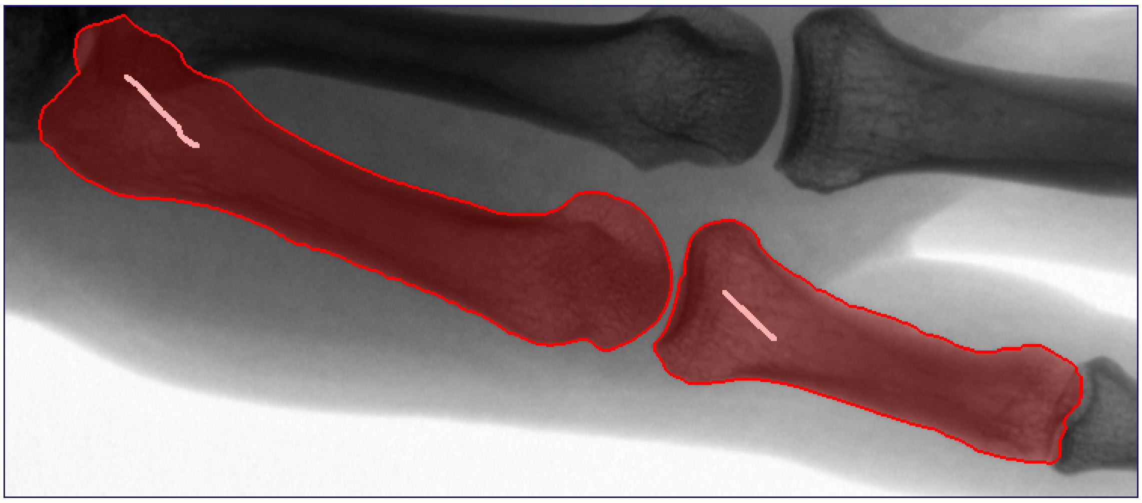


Figure 5.12: Two bones of the *Bone* image obtained using the pure geodesic energy.

Figure 5.14 shows the segmentation example of the artificial example already used in the previous section. One can see in Figure 5.14(c) that the segmentation relaying on the pure geodesic energy in this image badly fails. This is due to the very bad edge image as shown in Figure 5.14(b). The edge image does not contain any useful information on the true image borders. Thus heavy noise degrades the segmentation obtained by the pure geodesic energy as there is no useful edge information. A simple way to solve for this problem is to apply image denoising (see Figure 5.14(d)) before calculating the edges. Note that the edges in Figure 5.14(e) have significantly improved. In Figure 5.14(f) the desired segmentation is obtained after pre-filtering the image with our implementation of the ROF denoising algorithm. It is obviously very easy to find the desired segmentation. It showed that for the medical 3D datasets slight denoising before edge calculation, resulted in great improvements of the segmentation process.

In Figure 5.15 the *Kanizsa* image is segmented. No additional edges had to be introduced. The only used constraints were the foreground seed and the border set as background, as depicted in Figure 5.15(b). As one can see in Figure 5.15(c) the desired segmentation can be obtained using the segmentation approach relying on the pure geodesic energy. The foreground seedpoint have to be large enough, as otherwise the segmentation would not snap to the edges at the corners of the triangle. The approach using weak constraints approach would fail to deliver the desired segmentation, as there is no information in the gray values.

Another artificial segmentation example is presented in Figure 5.16. Here several

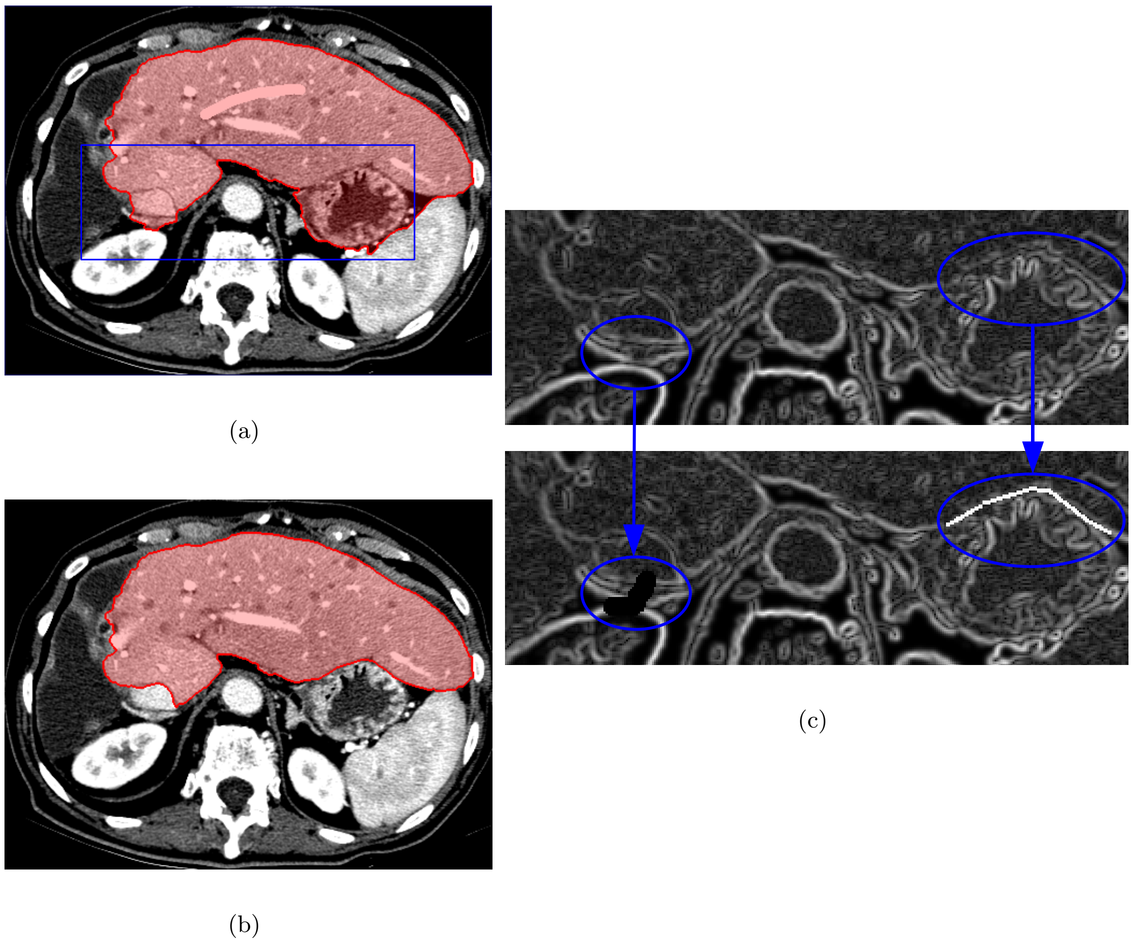


Figure 5.13: Segmentation process of a liver in a CT image. (a) The segmentation using only a foreground seed point and the border set to background. In the highlighted area the segmentation is wrong. (b) Modifying the edge information with the erase and the draw tool. (c) The final segmentation.

possible segmentations are possible. The chosen segmentation with the used constraints is depicted in Figure 5.16(a). It is obvious that in the middle ring another background constraint has to be induced. Otherwise the segmentations of the inner and outer ring would merge. This example should also show that the algorithm always tries to minimise the length of the border. The wide black rings always lead to edges at the inner and outer side. Our algorithm will always choose the inner one because of the smaller perimeter. In Figure 5.16(b) the edge borders are overlaid to the edge image to demonstrate this behaviour.

In 3D the segmentation process is inevitably more complex, and the algorithm clearly

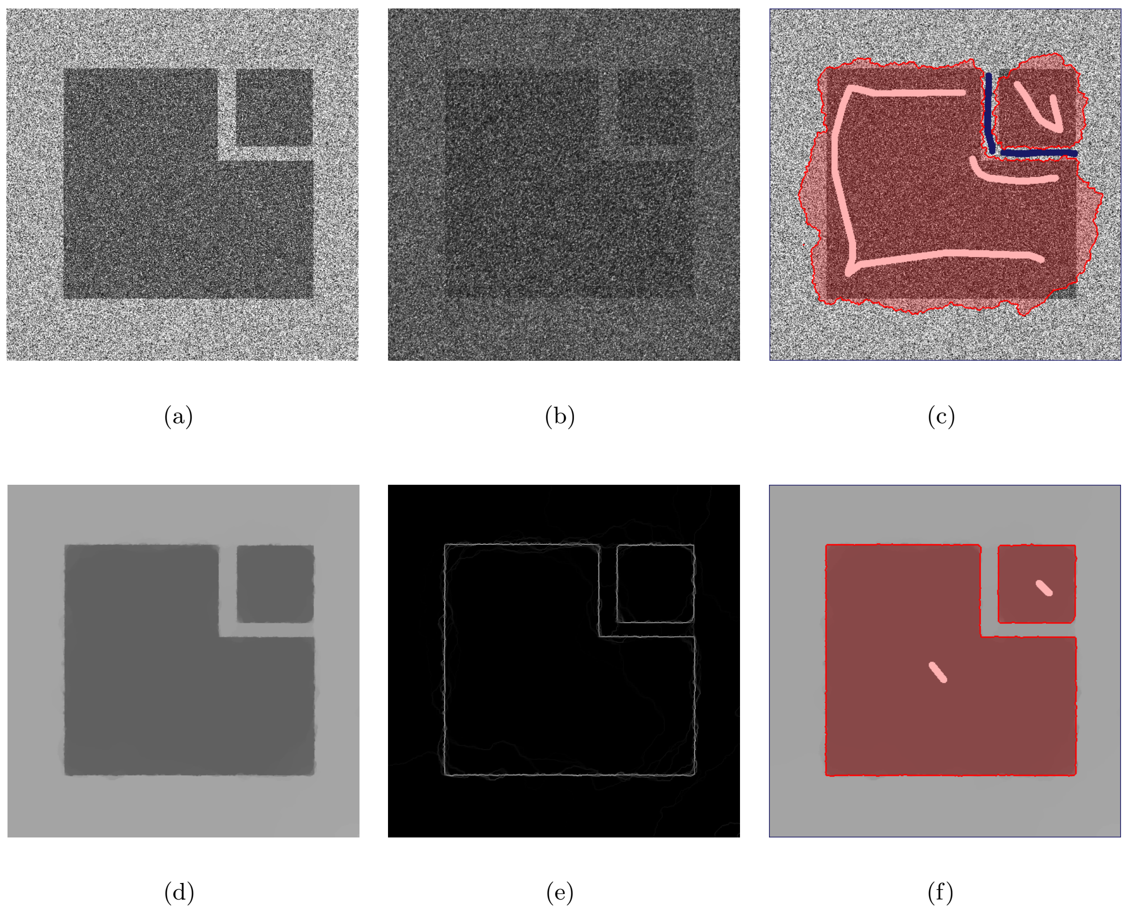


Figure 5.14: Segmentation using pure geodesic energy for an artificial example: (a) Original image, (b) edge image containing no useful information on desired borders, (c) failed segmentation, (d) ROF-denoised image, (e) new edge information, (f) the correctly segmented image.

needs more time to converge to the global minimiser. Nevertheless segmentations can be obtained in a very short time. The interactivity of the segmentation process is therefore not affected. Limited by the GPU memory of 768 MB the largest data set we were currently able to load has a size of $512 \times 512 \times 128$ voxels. Note that this is actually a large data set which can not be processed by most graph cut based approaches. The dataset from different views is depicted in Figure 5.17. Only a small gray value range is displayed to make it possible to view the viscera. In Figure 5.18 a $512 \times 512 \times 96$ CT image, that we will refer to as the *Abdomen* dataset, is segmented. The dataset has voxels of the size $0.55 \times 0.55 \times 2$ mm. The images in Figure 5.18 are screenshots of the segmentation process

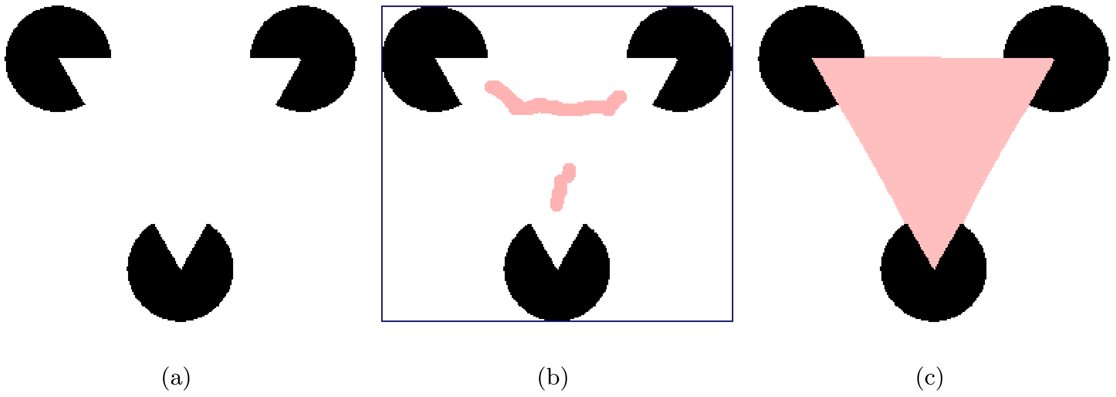


Figure 5.15: Segmentation of the *Kanizsa* image: (a) original image, (b) constraints set by the user and (c) the final segmentation.

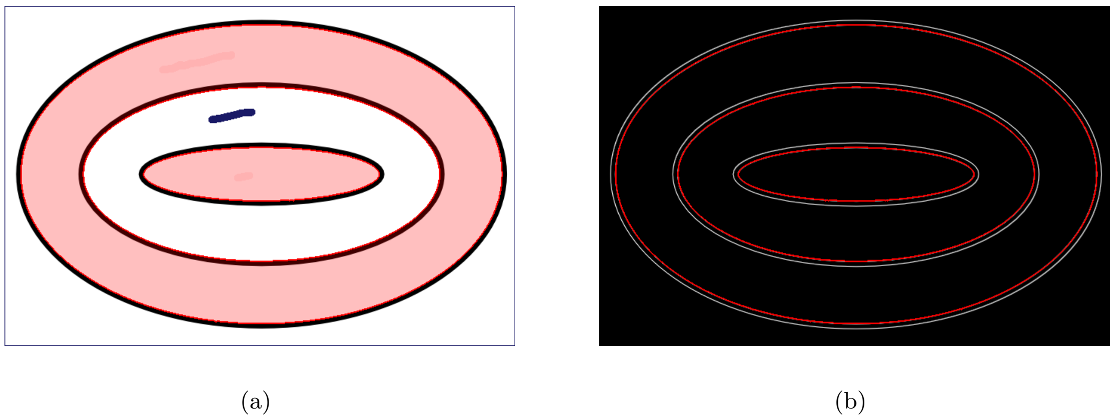


Figure 5.16: Another artificial segmentation example: (a) obtained segmentation with constraints and (b) the edge image with overlaid segmentation borders.

of the liver at different planes.

As the user starts to introduce constraints to the segmentation, the variable u will evolve towards the optimal solution. In 3D this evolution process can be watched, as speeds are not as fast as in the 2D version. Nevertheless the algorithms stays interactive all the time. Only for the initial segmentation (after drawing the first constraints) several seconds may be needed to find the optimal solution. When modifying the segmentation by introducing additional constraints, the algorithm reacts very fast even for large datasets. In Figure 5.19 the evolution process of the liver segmentation in the *Abdomen* dataset is depicted. The segmentation of Figure 5.19(a) shows the introduced foreground constraints, as this picture was taken immediately after reinitialising the variable u. After

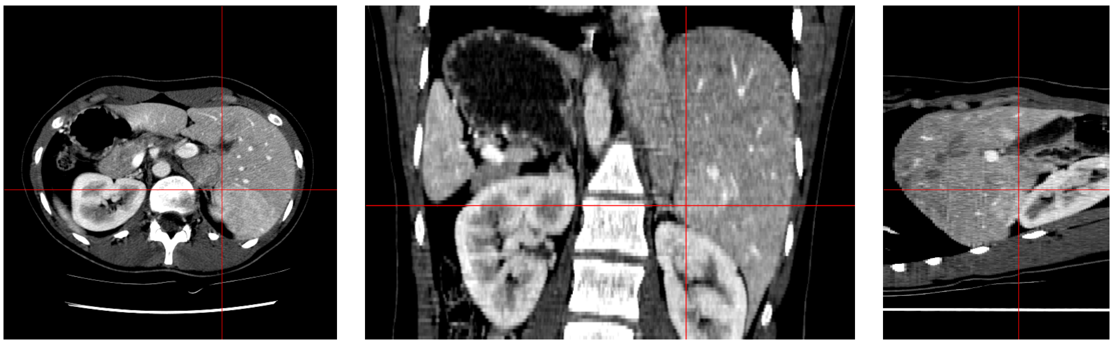

(a) (b) (c)

Figure 5.17: The *Abdomen* dataset: (a) axial view, (b) coronal view and (c) sagittal view.

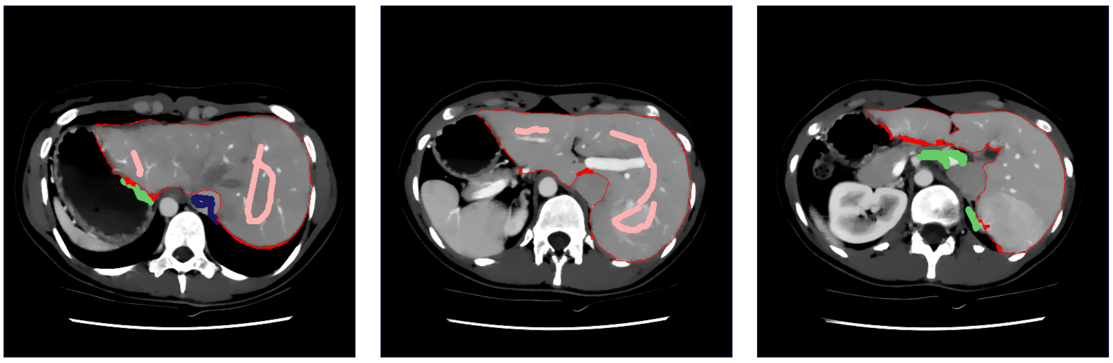

(a) (b) (c)

Figure 5.18: Segmentation process using local constraints in 3D: (a) plane 21, (b) plane 45 and (c) plane 57.

approximately 5 seconds, the segmentation already takes form as in Figure 5.19(b). After approximately a minute, the final segmentation as depicted in Figure 5.19(d), is reached.

In the *Abdomen* dataset we segmented the liver, the spleen and the kidneys. Figure 5.20 shows a 3D rendering of the segmented viscera. It took us about 8 minutes to obtain a segmentation of the liver, the kidneys and the spleen were obtained after approximately 5 minutes each. Using the implemented segmentation tool to extract the organs has shown to be a simple task.

In Figure 5.21 the segmentation of corpus callosum, cerebellum and brain stem in a brain MRI dataset is presented. The dataset is of the size $256 \times 320 \times 256$ and has voxels of the size $1 \times 1 \times 1$ mm. Gray values range from 1 to 233. In Figure 5.21(a - c) some slices

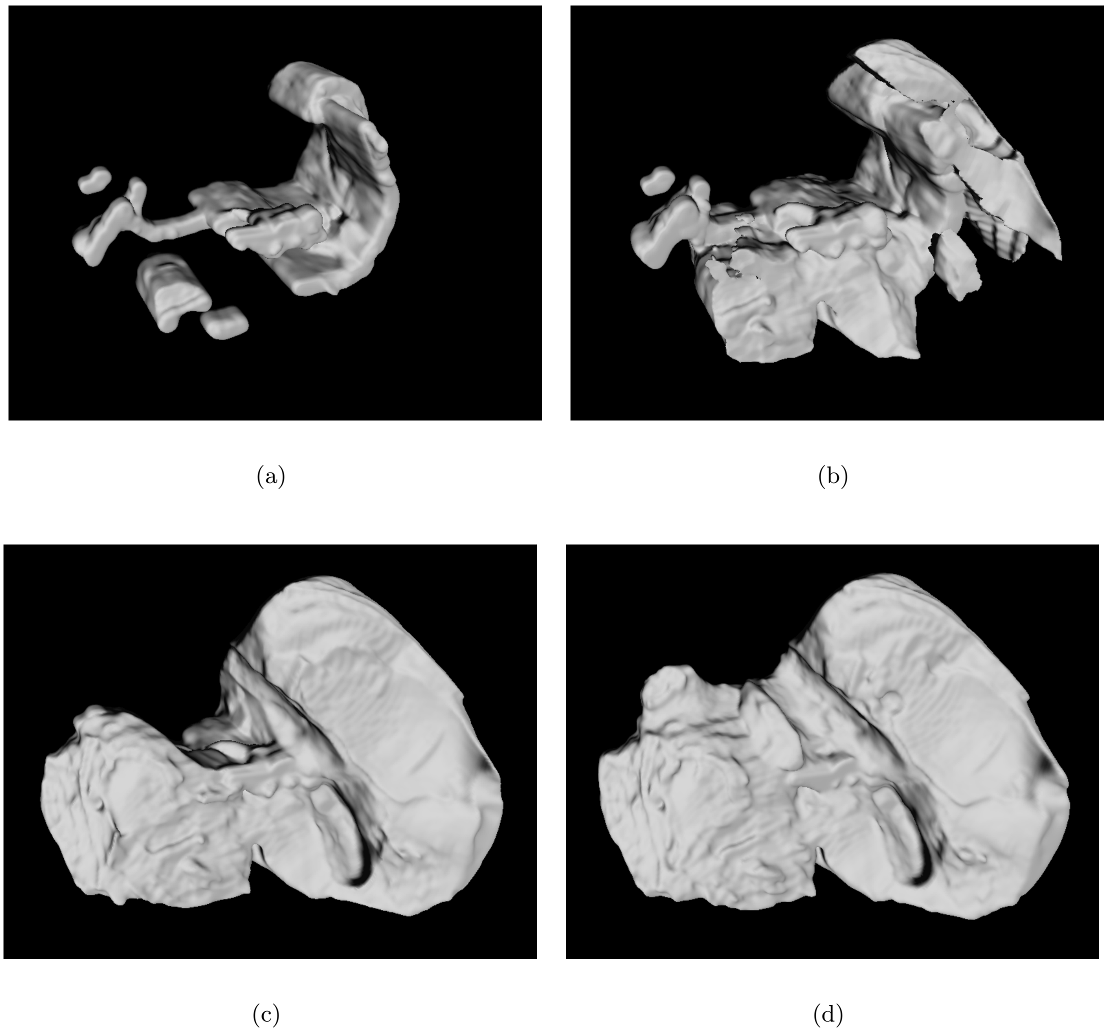


Figure 5.19: Curve evolution in 3D: (a) segmentation immediately after starting to iterate, (b) after only a few iterations calculated, (c) a more advanced intermediate example and (d) the final segmentation.

of the original dataset are presented. One can see that the cerebellum has a good contrast to the environment. Only at the transition to the brain stem no borders are present, as can be seen in the coronal view. This problem can be solved quickly after applying a few local constraints. The same applies to the segmentation of the brain stem. Segmentation of the corpus callosum proved a little bit more difficult, as in some parts edges are not well defined. Nevertheless approximately 5 minutes were needed for each of the segmentations. In Figure 5.21(d - e) renderings of the segmentations are depicted.

To present a more difficult example, we chose to segment a dataset of the knee in a

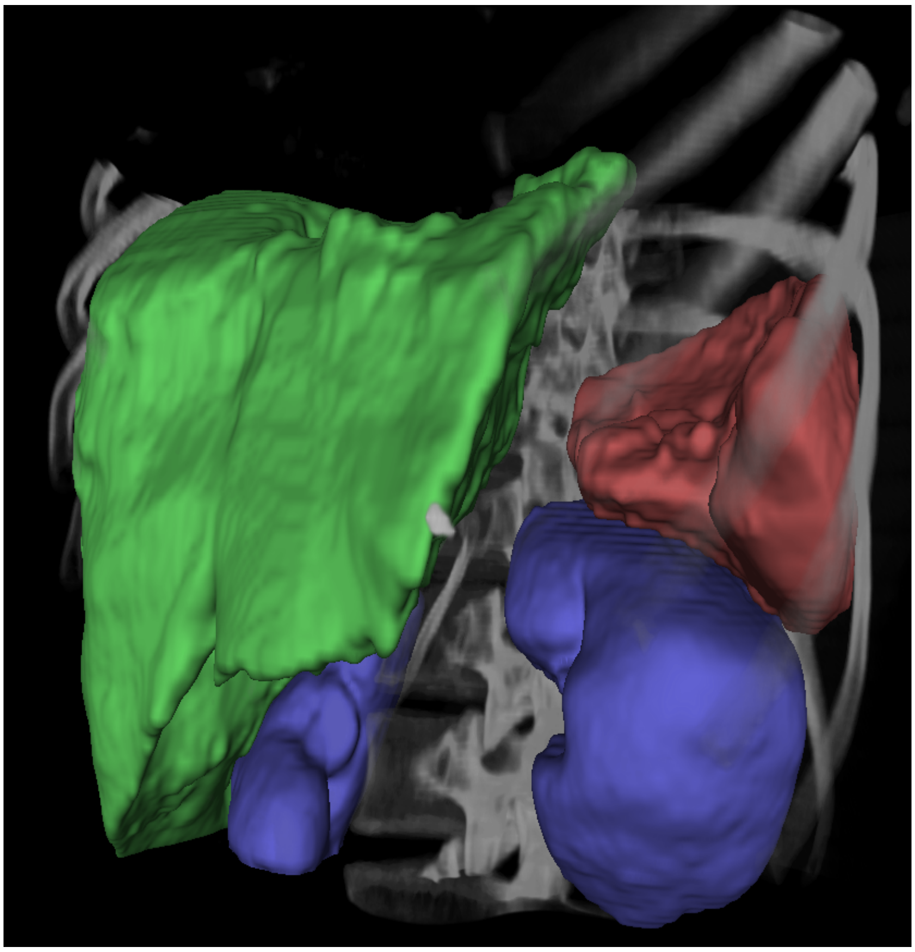
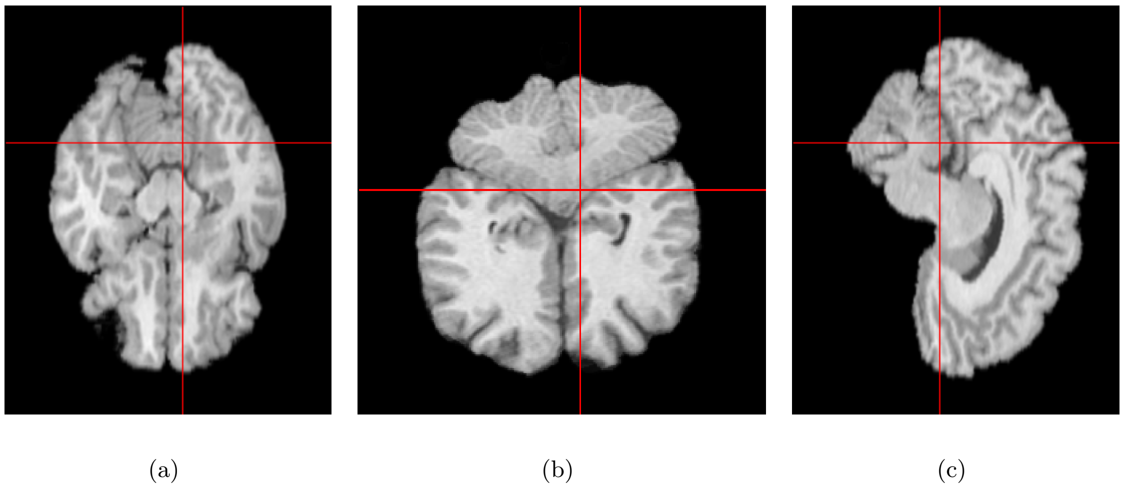


Figure 5.20: Segmentation of various viscera in the *Abdomen* dataset.

MRI dataset. Different views of the dataset are shown in Figure 5.22(a - c). The dataset originally has a size of $448 \times 512 \times 160$ with a voxel size of $0.293 \times 0.293 \times 0.6$ mm. Gray values range from 0 to 453. Due to our limitation of size caused by memory, the dataset was rescaled to the size $256 \times 256 \times 256$ to do the segmentation. Our goal was to segment the femur bone. It is easy to see that this data set has a low signal to noise ratio and the edges are very weak. Figure 5.22(d - e) shows 3D renderings of the segmented femur superimposed to a ortho-slice view of the original data set. To achieve this segmentation slightly more foreground seeds had to be set in weakly defined areas. Nevertheless only approximately 8 minutes were needed to get the desired segmentation.

As a final example in 3D, the segmentation of the lung in a CT dataset of a mouse is shown in Figure 5.23. The dataset was rescaled from originally $400 \times 400 \times 400$ to $256 \times 256 \times 256$. The voxel size of the original dataset is $0.056 \times 0.056 \times 0.056$. Gray



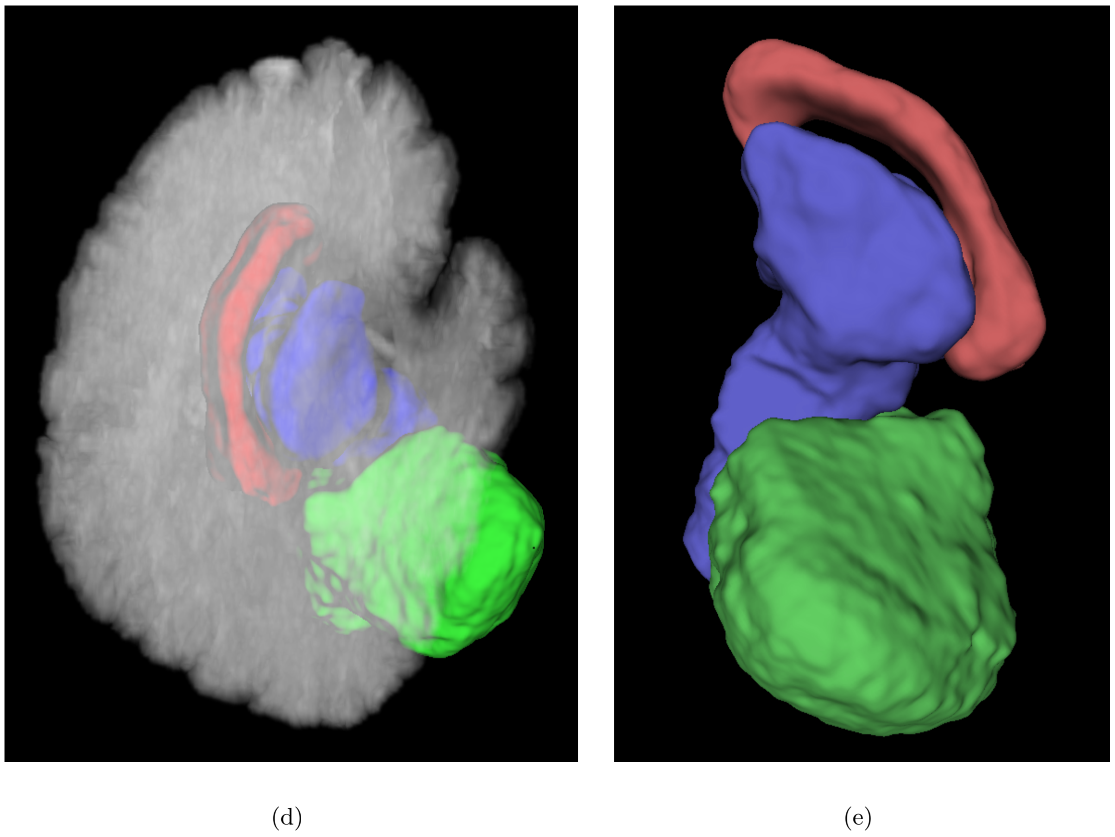


Figure 5.21: Segmentation of a brain MRI dataset. The original data in (a) axial view, (b) coronal view and (c) sagittal view. (d) and (e) show a renderings of the segmented corpus callosum, cerebellum and brain stem.

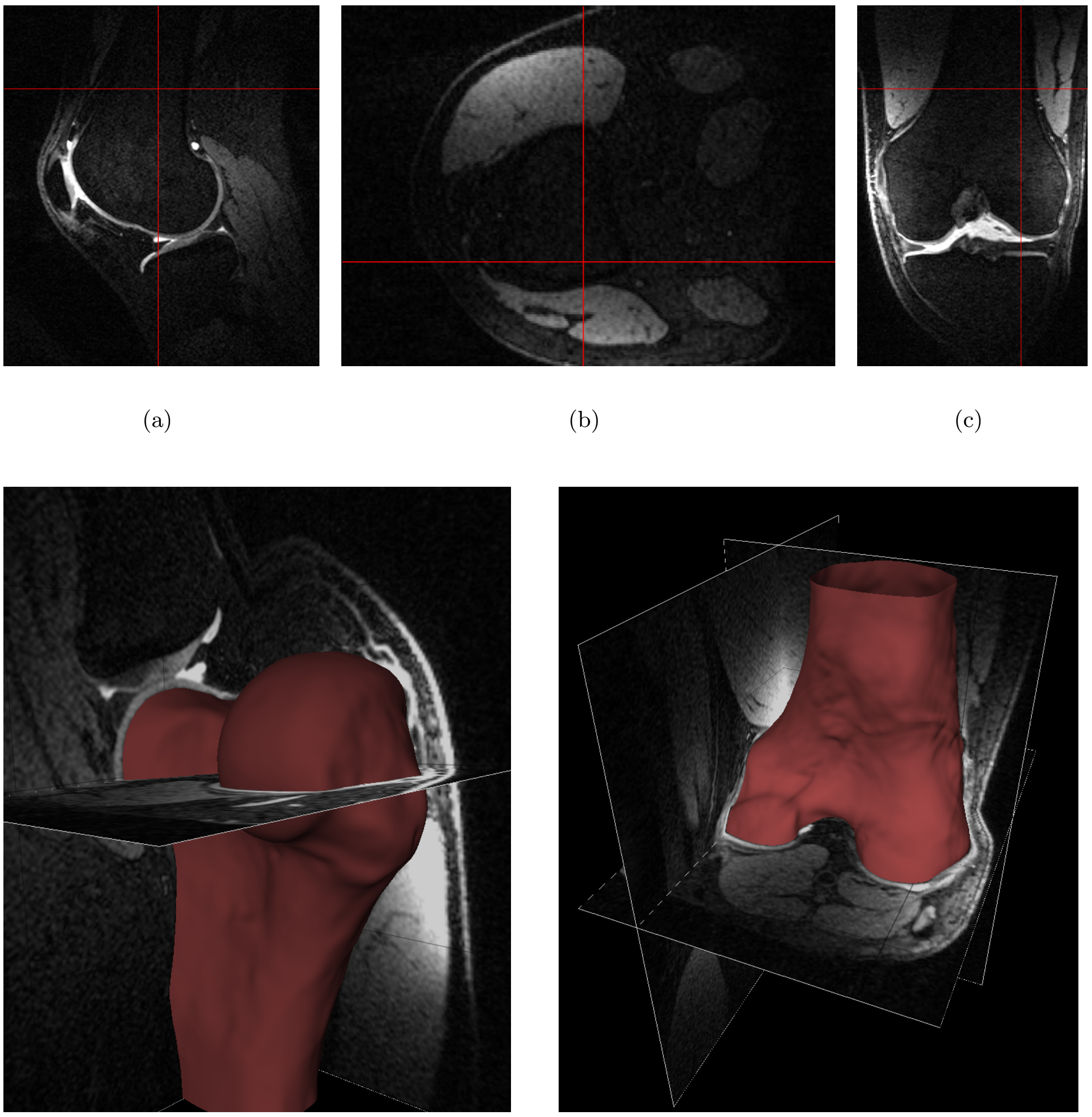
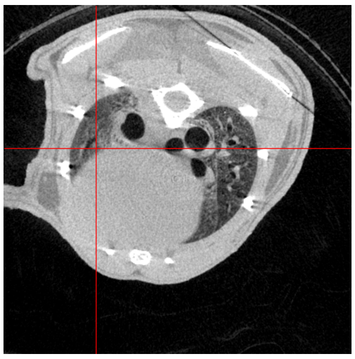
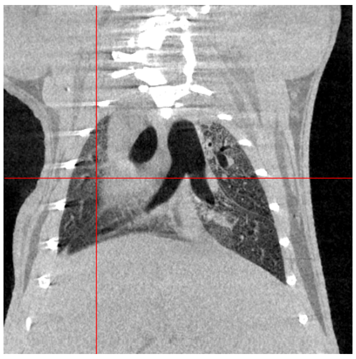


Figure 5.22: Segmentation of a knee MRI dataset. The original data in (a) sagittal view, (b) axial view and (c) coronal view. (d) and (e) show a rendering of the segmented femur.

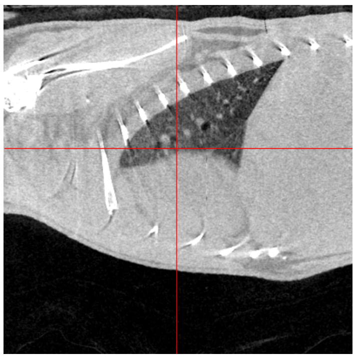
values are in the range from -2699 to 14416. For the pure geodesic energy approach the wide range of gray values in the lung represent no problems. A rendering of the final segmentation can be seen in Figure 5.23(d). Approximately 5 minutes were needed to obtain this segmentation.



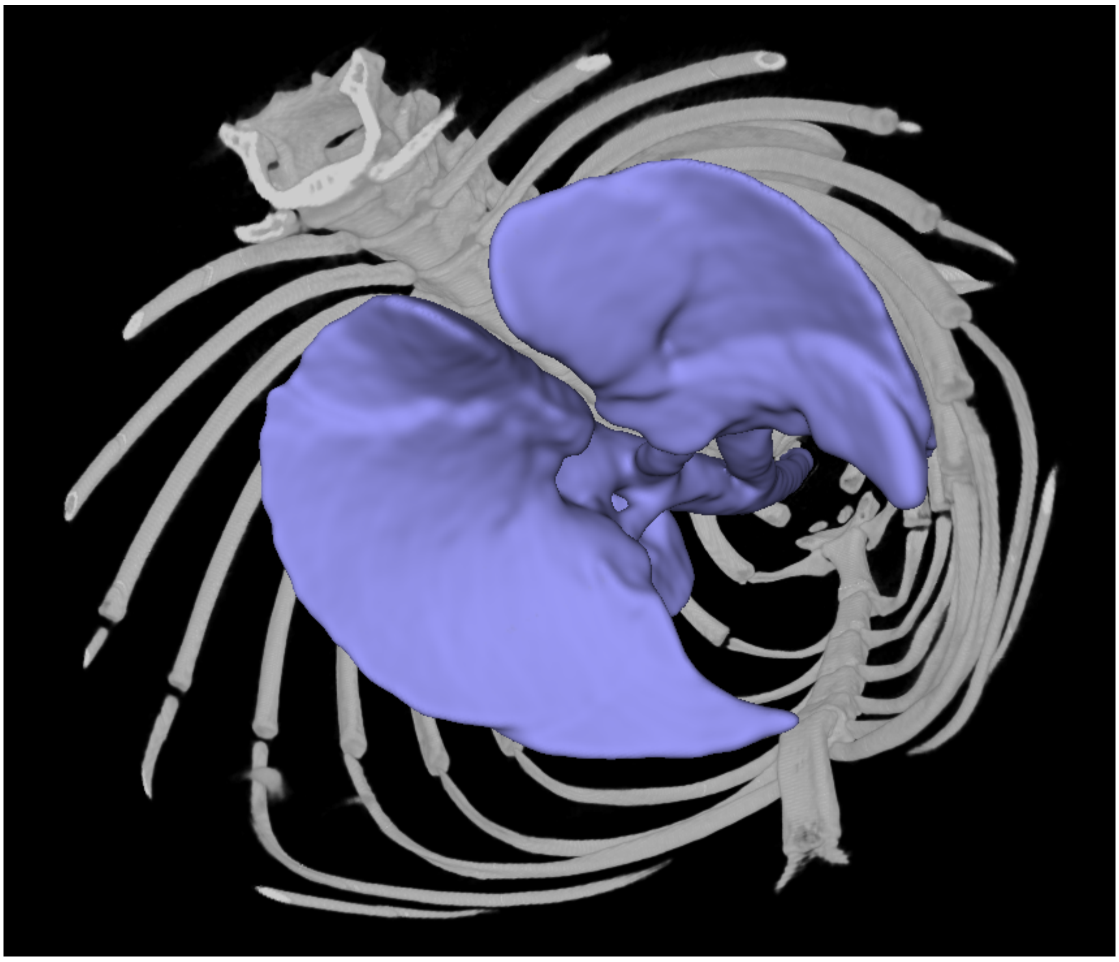
(a)



(b)



(c)



(d)

Figure 5.23: Segmentation of a CT lung dataset of a mouse. The original data in (a) axial view, (b) sagittal view and (c) coronal view. (d) and (e) show a rendering of the segmented lung.

We have already seen that the proposed algorithm works for a wide range of images. In Figure 5.24 segmentations of different other image modalities are shown. Figure 5.24(a) shows a microscopy image of blood cells of the size 1800 $\times$ 2239 pixels. Although this is relatively large for a 2D image, segmentation was very fast. Figure 5.24(b) and 5.24(c) were taken from [81]. In Figure 5.24(b) a broken bone in an arm was segmented in a x-ray image. Note that the contrast between bone and tissue was very low. Nevertheless our segmentation framework allowed to sufficiently segment the bone. In Figure 5.24(c) the hip bones were segmented in a nuclear image. Note that the signal to noise ratio was very small. Before we could segment this image, slight ROF denoising had to be applied. We did not set $\lambda = 0$ for this segmentation. Instead we used a combined approach with weak constraints, by setting λ to a very small value. Note that the segmentation would have vanished without any foreground seeds. In Figure 5.24(d) the four chambers of the heart are segmented in an ultrasound image. Ultrasound images usually suffer from strong speckles an weak borders. Consequently the segmentation of this image was very challenging. About a minute was needed to set all the background constraints, and to delete several edges. Note that we had to use background constraints to sufficiently separate the atria from the ventricles as the heart valves are only weakly defined and very thin. One can see that though sometimes a higher afford is needed, the proposed segmentation model can be applied to all medical image modalities.

5.3.3 Discussion

We have seen in the previous sections that the proposed segmentation model allows the segmentation of a wide range of images. We applied the segmentation model to x-ray, nuclear, microscopy, ultrasound, real world and artificial 2D images, and to CT and MRI datasets. The segmentation can be approached from two different directions. The segmentation approach based on weak constraints, uses the thresholding scheme to find an initial segmentation. By varying λ and applying additional local constraints, the desired globally optimal segmentation is obtained. This approach proved to be successful if the desired segmentation consists of homogeneous gray values. The second segmentation approach relies on the pure geodesic energy. Hard constraints have to be incorporated to find a meaningful segmentation. Solely the edge information is used to obtain the segmentation. This approach is therefore invariant to intensity inhomogeneities.

Due to the ambiguity of the segmentation task, user interaction is an important feature of a general purpose segmentation framework. By setting seed regions that can be

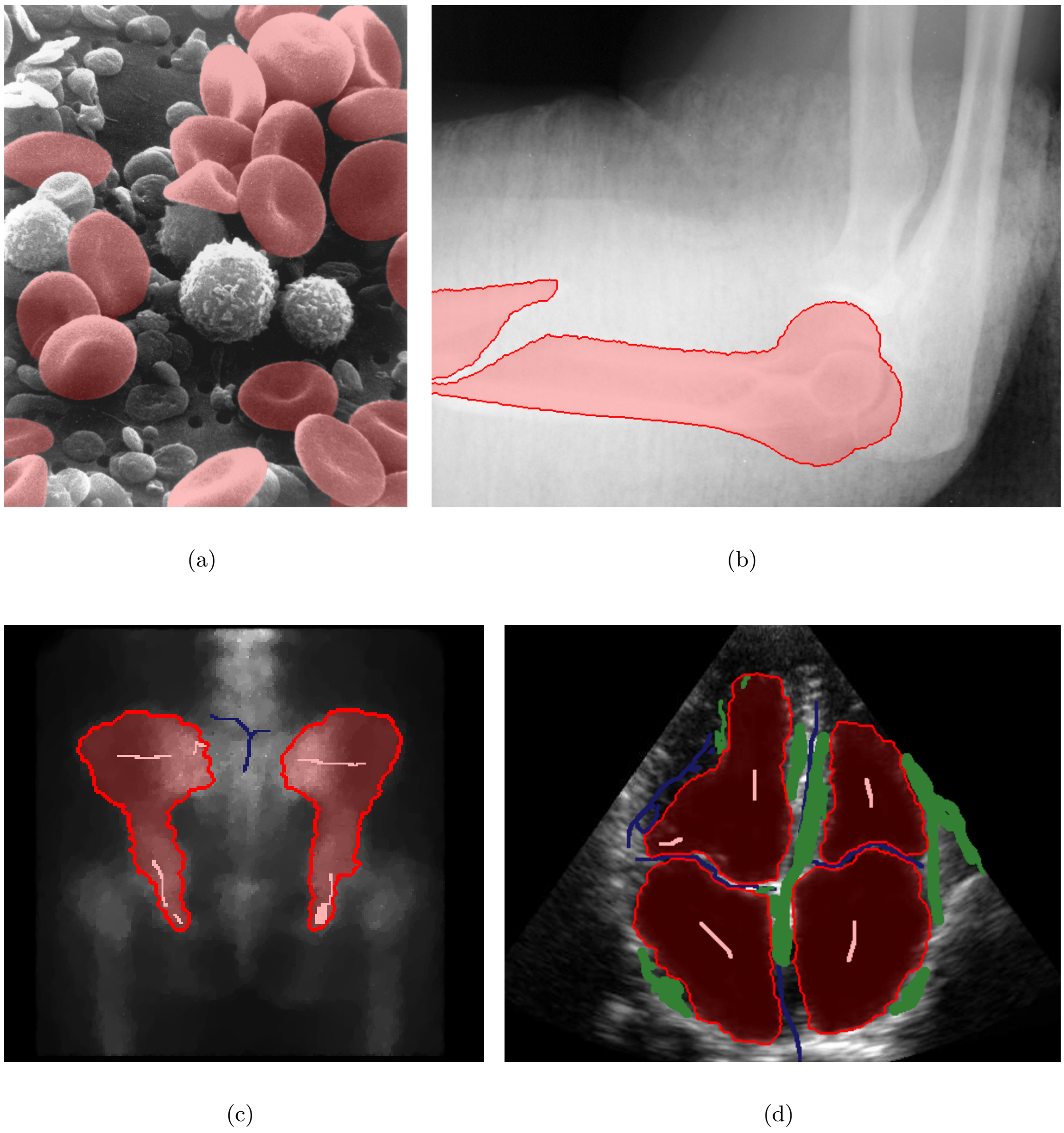


Figure 5.24: Segmentations of (a) a microscopy image of blood cells, (b) a x-ray image of a broken bone, (c) a nuclear image and (d) an ultrasound image of the heart.

either weak or hard constraints, the desired region in an image can be efficiently selected. It showed that the incorporation of additional local constraints is very effective for further refining the desired solution. Furthermore the edge modification tools proved to be successful for correcting incorrect contours.

As a conclusion we can remark that the segmentation approach using the pure geodesic energy seems to be better suited for 3D medical datasets than the approach using weak constraints. It showed that the implementation using CUDA is fast enough to make the algorithm real-time applicable even for large datasets. The only limiting factors seem to be the interface, and memory for large datasets. Clearly, a user interface working fully in 3D would be of great benefit. The work in [10] suggests some possible directions.

Chapter 6

Conclusion and Future Work

Contents

6.1 Conclusion

In this master's thesis we presented a fast interactive general purpose segmentation framework. We reviewed several common approaches to image segmentation in Section 1.4. It showed that most of the common algorithms are only applicable on a small class of segmentation problems, e.g. thresholding approaches work only on objects with a homogeneous gray value range.

Deformable models as presented in Section 2.1 are a promising approach to the segmentation task. Geodesic active contours and minimal surfaces by Caselles et al. [21, 22] present an intrinsic contour model that depends on geometrical information derived from the image. Unfortunately common approaches aiming to minimise the geodesic energy, e.g. level set methods, get stuck in local minima, and therefore need good initialisations. Graph based methods suffer from a metrication error that is induced by the definition of the grid. Large neighbourhood systems improve the segmentation results, but cause a strong increase in computational costs and memory consumption. The segmentation approach proposed in this master's thesis, guarantees to find a globally optimal solution to the GAC model, and does not suffer from any metrication errors.

The continuous maximum flow algorithm of Appleton and Talbot [4] was discussed in Section 2.2.4. The maximum flow algorithm finds the globally optimal solution of the

GAC model. In Section 3.1.3 we showed that the maximum flow algorithm is equal to a special case of our proposed segmentation algorithm. Other closely related work by Bresson et al. [15–17] and Leung and Osher [54] was discussed in Section 2.3.3.

We reviewed different image denoising models in Section 2.3. Though the standard denoising model of Rudin, Osher and Fatemi [76] is well suited for image denoising, it showed that a data fidelity term based on the L^1 norm can make the algorithm contrast invariant. This property proved very well when using the L^1 data fidelity term in our segmentation model.

The proposed segmentation approach is based on the g-weighted Total Variation. The energy contains a part that minimises the geodesic active contour model, and a data fidelity term that uses a L^1 norm. In Section 3.1 we discussed the properties of the proposed energy functional. It showed that our energy is non-strictly convex. We therefore can find more than one globally optimal solution. The resulting segmentation is defined as a continuous variable. By extracting a level set the final segmentation is obtained. There are two different approaches the proposed segmentation algorithm can be used:

- Weak constraints: By thresholding an initial segmentation (weak constraints) is obtained that can be further modified using shape denoising. Hard constraints can be incorporated to further improve the segmentation.

- Minimising the pure geodesic energy: Globally optimal geodesic active contours are computed. At least one foreground and one background constraint have to be set, to find a meaningful solution.

We successfully applied the segmentation model to all medical image modalities we introduced in Section 1.3. Experimental results in Section 5.3 showed that for most of the medical image segmentation tasks, the approach minimising the pure geodesic energy is superior the approach relying on weak constraints. On the other hand Section 5.3.1 shows that in some cases weak constraints deliver faster and better results. We point out that a good edge image is very important when using the minimisation of the pure geodesic energy. By using the implemented denoising algorithms before edge calculation we could significantly improve the segmentation process.

In Section 2.4 different algorithms for solving the denoising models were discussed. It showed that standard (primal) approaches need approximations to minimise the energy, and are very slow. Dual versions minimise a dual variable and are much faster. Experiments in Section 5.2 showed that the projected gradient descent algorithm [24] provides the fastest convergence times. In Section 3.1.2 we successfully applied the dual methods to the proposed segmentation energy.

We presented numerical methods used for implementation in Section 3.2. Our discretisation scheme leads only to a slight anisotropic behaviour of the algorithm.

The proposed segmentation algorithm is computationally quite expensive. On the other hand it is perfectly suited for parallelisation. Our implementation on the graphics device is very fast, and interactive even for large medical datasets. In Section 4.1 and 4.2 implementation specific topics were discussed. It showed that the CUDA framework together with current graphics hardware provides enough computational power for our algorithm. Memory management is one of the most critical parts of the implementation. We presented strategies to optimally utilise the device architecture. We also proposed a graphical user interface in Section 4.3.2, making it possible to easily incorporate high level knowledge. Experiments showed that the algorithm immediately reacts to new constraints induced by the user. The size of the datasets is only limited by the memory of the GPU.

We can conclude that our segmentation framework provides the user with a fast algorithm to interactively segment a wide range of images in two and three dimensions. The proposed algorithm delivers a globally optimal image segmentation considering local weak and hard constraints. The segmentation result can be arbitrarily refined using different brushes to incorporate high level information. Using fast numerical methods, and an implementation on the GPU our algorithm is highly interactive.

6.2 Future work

In future work we will concentrate on the simultaneous segmentation and rendering of the result by distributing the data over several GPUs. As volume rendering techniques and level set extraction in 3D are computationally very expensive, a separate GPU is necessary to keep the segmentation interactive. Furthermore large datasets could be distributed on several GPUs to account for the limitation of memory. We are also planning to improve the user interface for 3D applications, as the user interface was in most cases the time limiting factor of the segmentation.

The extension of the algorithm to colour images is another topic of future research. We will also try to improve the discretisation scheme, to address the slight anisotropic behaviour of our implementation.

Bibliography

[1] AMD (2005). Introduction to the DirectX 9 High Level Shading Language. Technical report, Advanced Micro Devices Inc., Sunnyvale, CA, USA.

[2] AMD (2006). ATI CTM Guide - Technical reference manual. Technical report, Advanced Micro Devices Inc., Sunnyvale, CA, USA.

[3] Amini, A., Weymouth, T., and Jain, R. (1990). Using dynamic programming for solving variational problems in vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(9):855–867.

[4] Appleton, B. and Talbot, H. (2006). Globally minimal surfaces by continuous maximal flows. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(1):106–118.

[5] Aubert, G. and Blanc Feraud, L. (1999). Some remarks on the equivalence between 2D and 3D classical snakes and geodesic active contours. *Intl. J. of Computer Vision*, 34(1):19–28.

[6] Aubert, G. and Kornprobst, P. (2006). *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations (second edition)*, volume 147 of *Applied Mathematical Sciences.* Springer-Verlag, New York, USA.

[7] Aujol, J.-F., Gilboa, G., Chan, T., and Osher, S. (2006). Structure-texture image decomposition–modeling, algorithms, and parameter selection. *Intl. J. of Computer Vision*, 67(1):111–136.

[8] Bellman, R. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NY, USA.

[9] Berkhin, P. (2002). Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA.

[10] Bornik, A., Beichel, R., Kruijff, E., Reitinger, B., and Schmalstieg, D. (2006). A hybrid user interface for manipulation of volumetric medical data. In *IEEE Symposium on 3D User Interfaces 2006 (3DUI 2006)*, Alexandria, USA.

[11] Boykov, Y. and Jolly, M. P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. volume 1, pages 105–112.

[12] Boykov, Y. and Kolmogorov, V. (2003). Computing geodesics and minimal surfaces via graph cuts. In *9th IEEE International Conference on Computer Vision (CVPR 2003)*, pages 26–33, Washington, DC, USA.

[13] Boykov, Y. and Kolmogorov, V. (Sept. 2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Trans. on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137.

[14] BrainWeb (2007). Simulated brain database. `http://www.bic.mni.mcgill.ca/brainweb/`. Visited on January 20th 2008.

[15] Bresson, X. (2005). *Image segmentation with variational active contours.* PhD thesis, EPFL, Lausanne.

[16] Bresson, X., Esedoglu, S., Vandergheynst, P., Thiran, J., and Osher, S. (2005). Global minimizers of the active contour/snake model. In *International Conference on Free Boundary Problems: Theory and Applications (FBP).*

[17] Bresson, X., Esedoglu, S., Vandergheynst, P., Thiran, J.-P., and Osher, S. (2007). Fast global minimization of the active contour/snake model. In *Journal of Mathematical Imaging and Vision*, volume 28, pages 151–167, Norwell, MA, USA.

[18] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. (2004). Brook for GPUs: stream computing on graphics hardware. In *ACM SIGGRAPH 2004*, pages 777–786, New York, NY, USA.

[19] Canny, J. (1986). A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence.*, 8(6):679–698.

[20] Carter, J. (2001). *Dual Methods for Total Variation-based Image Restoration.* PhD thesis, UCLA, Los Angeles, CA, USA.

[21] Caselles, V., Kimmel, R., and Sapiro, G. (1997a). Geodesic active contours. *Intl. J. of Computer Vision*, 22(1):61–79.

[22] Caselles, V., Kimmel, R., Sapiro, G., and Sbert, C. (1997b). Minimal surfaces: A three dimensional segmentation approach. *Nummer. Math.*, 77(4):423–451.

[23] Chambolle, A. (2004). An algorithm for total variation minimizations and applications. *Journal of Math. Imaging and Vision*, 20(1–2):89–97.

[24] Chambolle, A. (2005). Total variation minimization and a class of binary MRF models. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 136–152.

[25] Chambolle, A. and Lions, P.-L. (1997). Image recovery via total variation minimization and related problems. *Nummer. Math.*, 76(167-188).

[26] Chan, T. and Esedoglu, S. (2004). Aspects of total variation regularized L_1 function approximation. *SIAM Journal of Applied Mathematics*, 65(5):1817–1837.

[27] Chan, T., Esedoglu, S., Park, F., and Yip, A. (2005). *Mathematical Models in Computer Vision*, chapter Total Variation Image Restoration: Overview and Recent Developments. Springer.

[28] Chan, T., Golub, G., and Mulet, P. (1999). A nonlinear primal-dual method for total variation-based image restoration. *SIAM Journal of Applied Mathematics*, 20(6):1964–1977.

[29] Chan, T. and Vese, L. (2001). Active contours without edges. *IEEE Trans. Image Processing*, 10(2):266–277.

[30] Chaudhury, K. and Ramakrishnan, K. (2007). Stability and convergence of the level set method in computer vision. *Pattern Recognition Letters*, 28(7):884–893.

[31] Cohen, L. (1991). On active contour models and balloons. *Computer Vision, Graphics and Image Processing*, 53(2):211–218.

[32] Cohen, L. and Cohen, I. (Nov 1993). Finite-element methods for active contour models and balloons for 2-D and 3-D images. *Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147.

[33] Cohen, L., Cohen, I., and Ayache, N. (1992). Using deformable surfaces to segment 3-D images and infer differential structures. In *Proceedings 2nd European Conference on Computer Vision*, pages 648–652.

[34] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

[35] Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. (1995). Active shape models - their training and application. *Comput. Vis. Image Underst.*, 61(1):38–59.

[36] Donoser, M., Bischof, H., and Wiltsche, M. (2006). Color blob segmentation by MSER analysis. *IEEE International Conference on Image Processing*, pages 757–760.

[37] Enderle, J., Blanchard, S., and Bronzino, J., editors (2005). *Introduction to Biomedical Engineering Second Edition*. Elsevier Academic Press, London, UK.

[38] Fernando, R. and Kilgard, M. J. (2003). *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[39] Funka-Lea, G., Boykov, Y., Florin, C., Jolly, M.-P., Moreau-Gobard, R., Ramaraj, R., and Rinck, D. (2006). Automatic heart isolation for CT coronary visualization using graph-cuts. *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pages 614–617.

[40] Goldfarb, D. and Yin, W. (2007). Parametric maximum flow algorithms for fast total variation minimization. Technical report, Rice University.

[41] Gouttard, S., Styner, M., Joshi, S., Smith, R., Hazlett, H., and Gerig, G. (2007). Subcortical structure segmentation using probabilistic atlas priors. In *Proceedings of MICCAI 2007*, pages 37–46. Presented at Worshop - 3D Segmentation in the Clinic: a Grand Challenge.

[42] Grady, L. (2004). *Space-Variant Computer Vision: A Graph-Theoretic Approach*. PhD thesis, Boston University, Boston, MA.

[43] Grady, L. (2006). Random walks for image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783.

[44] Grady, L., Sun, Y., and Williams, J. (2006). Interactive graph-based segmentation methods in cardiovascular imaging. In Paragios, N., Chen, Y., and Faugeras, O., editors, *Handbook of Mathematical Models in Computer Vision*, pages 453–469. Springer.

[45] Hornak, J. (1999). The basics of MRI. `http://www.cis.rit.edu/htbooks/mri/index.html`. Visited on January 20th 2008.

[46] Horowitz, S. and Pavlidis, T. (1974). Picture segmentation by a directed split and merge procedure. In *Procedings of the 2nd International Joint Conference on Pattern Recognition*, pages 424–433.

[47] Hough, P. (1962). Method and means for recognizing complex patterns. In *US Patent 3,069,654*.

[48] Huang, J. and Mumford, D. (1999). Statistics of natural images and models. *IEEE Conference on Computer Vision and Pattern Recognition*, 1:541–547.

[49] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

[50] Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *Intl. J. of Computer Vision*, 1(4):321–331.

[51] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A., and Yezzi, A. (1995). Gradient flows and geometric active contour models. In *Fifth International Conference on Computer Vision*, pages 810–815.

[52] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A., and Yezzi, A. (1996). Conformal curvature flows: From phase transitions to active vision. *Archive for Rational Mechanics and Analysis*, 134:275–301.

[53] Lakare, S. (2000). 3D segmentation techniques for medical volumes. Technical report, State University of New York at Stony Brook.

[54] Leung, S. and Osher, S. (2005). Fast global minimization of the active contour model with TV-inpainting and two-phase denoising. In *3rd IEEE Workshop Variational, Geometric, and Level Set Methods in Computer Vision*, pages 149–160.

[55] Li, C., Xu, C., Gui, C., and Fox, M. (2005). Level set evolution without re-initialization: a new variational formulation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:430–436.

[56] Lin, P., Yan, X., Zheng, C., and Yang, Y. (2004). Medical image segmentation based on mumford-shah model. *International Conference on Communications, Circuits and Systems*, 2:942–945.

[57] Maintz, J. and Viergever, M. (1998). A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36.

[58] Malladi, R., Sethian, J. A., and Vemuri, B. C. (1995). Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175.

[59] Mayer, A. and Greenspan, H. (2006). Segmentation of brain MRI by adaptive mean shift. In *IEEE International Symposium on Biomedical Imaging*, pages 319–322.

[60] McCool, M. and Toit, S. D. (2004). *Metaprogramming GPUs with Sh.* AK Peters Ltd.

[61] McInerney, T. and Terzopoulos, D. (1995). A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4D image analysis. *Comp. Med. Imag. Graph.*, 19(1):69–83.

[62] McInerney, T. and Terzopoulos, D. (1996). Deformable models in medical image analysis: A survey. *Medical Image Analysis*, 1(2):91–108.

[63] Mumford, D. and Shah, J. (1985). Boundary detection by minimizing functionals. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 22–26.

[64] Mumford, D. and Shah, J. (1988). Optimal approximations by piecewise smooth functions and variational problems. *Communications on Pure and Applied Mathematics*, XLII(5):577–685.

[65] Nikolova, M. (2004). A variational approach to remove outliers and impulse noise. *Journal of Mathematical Imaging and Vision*, 20(1-2):99–120.

[66] Nikolova, M., Esedoglu, S., and Chan, T. (2006). Algorithms for finding global minimizers of image segmentation and denoising models. *SIAM Journal of Applied Mathematics*, 66(5):1632–1648.

[67] Noble, J. A. and Boukerroui, D. (2006). Ultrasound image segmentation: A survey. *IEEE Transactions on Medical Imaging*, 25(8):987–1010.

[68] NVidia (2006). NVidia GeForce 8800 GPU architecture overview. Technical report, NVIDA Corp., Santa Clara, CA, USA.

[69] NVidia (2007). NVidia CUDA Compute Unified Device Architecture programming guide 1.1. Technical report, NVIDA Corp., Santa Clara, CA, USA.

[70] Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *J. Comput. Phys.*, 79(1):12–49.

[71] Pham, D., Xu, C., and Prince, J. (2000). Current methods in medical image segmentation. *Annu. Rev. Biomed. Eng.*, 2.

[72] Pock, T., Grabner, M., and Bischof, H. (2007). Real-time computation of variational methods on graphics hardware. In *Computer Vision Winter Workshop 2007*. Institute for Computer Graphics and Vision, Graz University of Technology.

[73] RapidMind (2008). Rapidmind Inc. `http://www.rapidmind.net`. Visited on January 18th 2008.

[74] Rohlfing, T., Brandt, R., Menzel, R., Russakoff, D. B., and Maurer, Jr., C. R. (2005). Quo vadis, atlas-based segmentation? In Suri, J., Wilson, D. L., and Laxminarayan, S., editors, *The Handbook of Medical Image Analysis – Volume III: Registration Models*, chapter 11, pages 435–486. Kluwer Academic / Plenum Publishers, New York, NY.

[75] Rost, R. J. (2006). *OpenGL Shading Language (2nd Edition)*. Addison-Wesley Professional.

[76] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Phys. D*, 60(1-4):259–268.

[77] Sethian, J. A. (1999). *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry*. Cambridge University Press, Cambridge.

[78] Sezgin, M. and Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–168.

[79] Shallows (2008). Shallows, making GPGPU programming fast and easy. `http://shallows.sourceforge.net/`. Visited on January 18th 2008.

[80] Sinop, A. K. and Grady, L. (2007). A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *Proc. of ICCV 2007*. IEEE Computer Society, IEEE.

[81] Snyder, W. E. (2002). NC state university image analysis laboratory database. `http://www.ece.ncsu.edu/imaging/Archives/ImageDatabase/index.html`. Visited on January 21th 2008.

[82] Sonka, M. and Fitzpatrick, J. M., editors (2000). *Handbook of Medical Imaging - Volume 2. Medical Image Processing and Analysis*. SPIE Press, Bellingham, Washington, USA.

[83] Sonka, M., Hlavac, V., and Boyle, R. (2007). *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering.

[84] Suri, J. S., Setarehdan, S. K., and Singh, S., editors (2002). *Advanced algorithmic approaches to medical image segmentation: state-of-the-art application in cardiology, neurology, mammography and pathology*. Springer-Verlag, NY, USA.

[85] Tarditi, D., Puri, S., and Oglesby, J. (2006). Accelerator: using data parallelism to program GPUs for general-purpose uses. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 325–335, NY, USA.

[86] Terzopoulos, D., Witkin, A., and Kass, M. (1988). Constraints on deformable models: Recovering 3D shape and nonrigid motion. *Artificial Intelligence*, 36(1):91–123.

[87] Unger, M., Pock, T., and Bischof, H. (2008). Continuous Globally Optimal Image Segmentation with Local Constraints. In *Proc. of Computer Vision Workshop 2008*. Slovenian Pattern Recognitian Society.

[88] Vincent, L. and Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583–598.

[89] Vogel, C. and Oman, M. (1996). Iteration methods for total variation denoising. *SIAM Journal of Applied Mathematics.*, 17:227–238.

[90] Williams, D. and Shah, M. (1990). A fast algorithm for active contours. In *Proceedings 3rd International Conference on Computer Vision*, pages 592–595.

[91] Xu, R. and Wunsch, I. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678.