

Thesis to obtain a Master degree at the Shanghai Jiaotong University

Liver Perfusion using Level Set Methods

Author	Sebastian Nowozin
Supervisor	Professor Gu Lixu
Research direction	Image Guided Surgery and Therapy Laboratory
Document date	July 18, 2005

I certify the following work to be my own creation, except where explicitly stated otherwise. In the later case the references to the original sources are provided.

to my parents, Annette and Reinhard

Contents

1	Introduction	9
1.1	Motivation and clinical challenge	9
1.1.1	Liver perfusion series MRI data	9
1.2	Main contribution	10
1.3	Previous work	10
1.4	Outline	11
1.5	Acknowledgments	11
2	Segmentation	13
2.1	Introduction	13
2.1.1	Definition	13
2.1.2	Properties of segmentation algorithms.	15
2.1.3	Applications of Medical Image Segmentation	17
2.2	Medical Image Segmentation algorithms overview	18
2.3	Thresholding based segmentation methods	21
2.3.1	Basic thresholding	21
2.3.2	Adaptive thresholding	23
2.3.3	Local thresholding	25
2.4	Region based segmentation methods	25
2.4.1	Region growing	25
2.4.2	Region splitting and merging	26
2.5	Watershed segmentation	26
2.6	Snakes	29
2.6.1	Parametrization methods	30
2.6.2	Problems of the original Snake model	30
2.6.3	Proposed solutions	31
2.6.4	Conclusion	32
3	The Level Set Method	33
3.1	Interface evolution	33
3.1.1	Traditional technique: markers	34
3.1.2	Traditional technique: Volume-of-fluid	36
3.2	Level Set Method	37
3.2.1	Implicit representation of an interface	37
3.2.2	The signed distance function	38
3.2.3	Geometric properties	40

3.3	Ingredients	44
3.3.1	Initialization/Reinitialization methods	44
3.3.2	Evolution methods	47
3.3.3	HJ ENO scheme	51
3.3.4	Isocontour extraction	53
3.4	The narrow-band level set method	54
3.5	Implementation concerns	56
3.5.1	Discretization	56
3.5.2	Handling the boundary	57
3.5.3	Reinitialization frequency	57
4	The Fast Marching Method	59
4.1	Comparison to the Level Set Method	59
4.2	The method	60
4.2.1	Marching step	60
4.2.2	Update rule	63
4.2.3	Heap details	66
4.3	Implementation	68
5	Liver Perfusion - Approach	69
5.1	Overview	69
5.2	MRI - Magnetic Resonance Imaging	70
5.2.1	MRI data modality	71
5.2.2	Employed data format	71
5.3	Segmenting the liver	71
5.3.1	Locating the seed point	71
5.3.2	FMM Segmentation Step	72
5.3.3	Level set segmentation step	75
5.4	Perfusion Area Localization	80
5.4.1	Distance Vector Transform	80
6	Liver Perfusion - Experiments and Results	83
6.1	Prototype implementation	83
6.1.1	Graphical User Interface	84
6.2	Experiments	86
6.2.1	Parameters	87
6.3	Results and discussion	87
6.3.1	Clinical evaluation	87
6.3.2	Performance evaluation	88
6.3.3	Segmentation accuracy evaluation	88
6.3.4	Perfusion area location accuracy evaluation	88
7	Conclusions	91
7.1	Future work	91

List of Figures

1.1	MRI example image	10
2.1	Example 2D gray scale image	16
2.2	Medical image segmentation taxonomy tree	18
2.3	Example bubble evolution	20
2.4	Gray scale image of objects to be segmented.	21
2.5	Image 2.4 thresholded at $T = 94$	21
2.6	Image 2.4 thresholded at $T = 109$	21
2.7	Image 2.4 thresholded at $T = 167$	21
2.8	Intensity histogram of 2.4	22
2.9	Gray scale image of objects to be segmented.	23
2.10	OTSU algorithm, $T = 125$	23
2.11	Entropy method, $T = 121$	23
2.12	Isodata algorithm, $T = 124$	23
2.13	Region splitting scheme on a quad-tree structure.	26
2.14	Blobs to be segmented.	28
2.15	Gaussian smoothed image 2.14, $\sigma = 3.0$	28
2.16	Segmented watershed/divide lines.	28
2.17	Watershed catchment basins.	28
2.18	Normalized gradient magnitude image of image 2.14.	28
2.19	Gaussian smoothed image 2.18, $\sigma = 3.0$	28
2.20	Segmented watershed/divide lines.	28
2.21	Watershed catchment basins.	28
3.1	Example curve represented by markers.	34
3.2	Swallow-tailing problem in the marker method	36
3.3	Volume-of-fluid method	37
3.4	A curve represented implicitly by a function.	38
3.5	Embedding a circle in a signed distance function ϕ	39
3.6	Zero isocurve of figure 3.5	39
3.7	Osculating circle and curvature	42
3.8	Discrete approximations to the Laplacian	43
3.9	KNOWN sets from zero crossing elements	47
3.10	Narrowband Level Set Method	55
4.1	FMM Step 1	62
4.2	FMM Step 2	62

4.3	FMM Step 3	63
4.4	FMM Step 4	63
4.5	Choosing quadrants for 2D FMM	63
5.1	Approach to liver perfusion measurement	69
5.2	Three different Gaussian speed modifier functions.	73
5.3	Result of the FMM segmentation step.	74
5.4	Liver region inside image 5.3.	74
5.5	The balance between diffusion and reaction of the speed function F_c	77
5.6	Result of the level set segmentation step.	79
5.7	Liver region inside image 5.6.	79
5.8	Defining a point using a distance vector	81
6.1	Prototype GUI: Initialization	84
6.2	Prototype GUI: Perfusion plot	85
6.3	Typical liver perfusion intensity curve	87
6.4	Comparison: manual and automatic marking of the liver boundary	89

Chapter 1

Introduction

In this chapter I introduce the scope and context of the thesis to you. Motivated by a real world clinical challenge I give a list of objectives in order to solve the challenge. To provide you, the reader, an orientation for the thesis, every chapter is summarized in an outline.

1.1 Motivation and clinical challenge

The problem to be solved with my thesis occurs regularly at the Shanghai First People Hospital. Dr. Zhang Hao, Mr. Xie Xueqian, Ms. Guai Hua and Mr. Zhang Tiannin introduced the problem to Prof. Gu Lixu and me. For certain illnesses related to the liver the blood flow to the liver has to be studied. By injecting a contrast agent into the patients arm while continuously taking MRI images, the concentration of the contrast agent can be studied while it flows through the patients body. A short time after the injection it reaches the liver and the MRI images at that time give important information about the blood supply of the liver.

To study the concentration of the agent in the liver for the entire series, the doctor currently has to manually analyze the images. Because the liver moves vertically as the patient breathes throughout the series, the perfusion-relevant position in the image moves as well¹. The doctor has two choices, a) to ignore the change in position and b) the mark the positions manually in all the images. After all positions are marked using one of the two methods, the intensities of the MRI images at these positions are plotted over time and the concentration curve of the contrast agent is deduced. The curve is used for further diagnosis.

Ignoring the change in position of the perfusion area introduces a measurement error. However, while the process of manually marking all the images leads to good results, it is very time consuming. The goal of this thesis is to automate the process so that only minimal manual work is required from the doctor.

1.1.1 Liver perfusion series MRI data

Thirteen MRI perfusion series were given to us for experiments. They show the patients abdomen in regular time intervals during the perfusion studies. An example set of four images taken from the series is displayed in figure 1.1.

¹For our series, the portal vein is used as perfusion area.

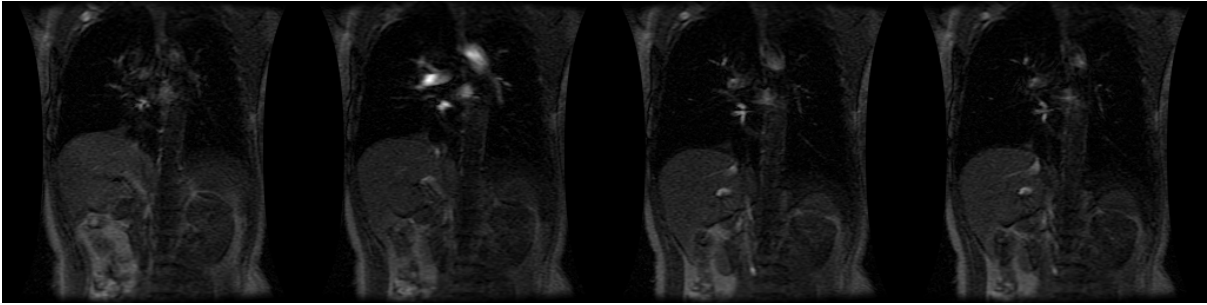


Figure 1.1: Representative example images from one of the provided MRI image series.

1.2 Main contribution

The contribution of this thesis is a solution to the clinical challenge of liver perfusion measurements. Specifically, the following has been achieved.

1. Robust, automatic and performant segmentation of the provided liver perfusion data sets.

Robust means the resulting procedure shall work on all the provided image series and will most definitely also work on new series. *Automatic* means the procedure shall not require manual steps with the exception of an initialization step. *Performant* means the procedure must work within acceptable time constraints on common Personal Computers.

2. Provide automatic intensity curve of a given subarea of the liver across the entire image series.

The valuable result in a clinical environment is the concentration time curve of a certain area within the liver. This concentration curve can directly be deduced from the image intensity. The procedure developed provides this intensity curve for a given area of the liver. As the patient breathes the liver moves, and the procedure provides means of image registration and transformation between the images in the series.

3. Evaluate the results of different parameters to the level set method.

1.3 Previous work

In the literature I have not found any attempt to solve the problem of liver perfusion measurement using level set methods.

Because my proposed approach is closely related to medical image segmentation and registration, for which there is a large body of literature available, I give detailed accounts of previous work in their respective chapters, 2, 3, 4 and 5. I find it important to provide references *in context*, instead of just providing them in listed form.

1.4 Outline

In the following chapters I describe all the employed methods in our approach, the approach itself and an evaluation of the experimental results.

In **chapter 2** we give an introduction to segmentation of digital images. The problems of segmentation in general are detailed and explained using an example image. Afterwards we list desirable properties of medical image segmentation algorithms and point to possible applications. The remaining part of the chapter concentrates on specific medical image segmentation algorithms, which we first introduce by giving a hierarchical overview and later by explaining the most important algorithms in detail.

Chapter 3 explains the level set method and its implementation. The chapter is self-contained and can be read separately. The level set method is introduced as an interface evolution algorithm and contrasted with two other common evolution methods, the marker method and the volume-of-fluid method. The representation the level set method uses is examined in detail and its importance as the fundamental part of the method is highlighted. The remaining chapter deals with the concrete problems that have to be solved when implementing the method, namely reinitialization, evolution and higher order approximations. Finally, details to the narrow-band level set method and implementation advice are given.

Chapter 4 explains the “cousin of the level set method”, the Fast Marching Method. Like the previous chapter, it is also self-contained. The method is first contrasted with the Level Set Method and in the following details and pseudo code are given.

Throughout **chapter 5** we describe our solution to the liver perfusion problem based on the theoretical base of chapters 2 to 4. The solution consists of three basic steps: segmentation, registration and measurement. The segmentation is the most complex part and described in detail. To solve the registration problem a simple novel scheme is devised and explained. Finally the measurement part and the relationship of the measurement results to the clinical application are described.

Chapter 6 details the prototype implementation and the results of applying the approach on real perfusion series. First the features of the prototype are described and the available options in graphical user interface (GUI) are explained. Afterwards, the segmentation accuracy and runtime performance of the prototype are examined and its clinical value established.

Concluding, in **chapter 7** I give an overview of what has been achieved throughout the thesis, what practical implications it has for possible clinical use and how the work might be continued in the future to improve upon the results.

1.5 Acknowledgments

I would like to thank Professor Gu Lixu for supervising this thesis and the insightful comments he provided throughout the whole process. For their continuous efforts in the exchange student program this thesis is part of I would like to thank Professor Li Fang and Professor Sheng Huanye from the Shanghai Jiaotong University and Professor Günter Hommel from the Technical University of Berlin. Dr. Zhang Hao, Mr. Xueqian Xie, Ms. Guai Hua and Mr. Zhang explained the clinical problem to me in detail and I am grateful to them.

Chapter 2

Segmentation

2.1 Introduction

We now introduce the problem of segmentation for digital images. First, we give a definition and derive criteria a good segmentation algorithm should possess. Then, we give an overview of segmentation in the field of medical imaging and later explain the most common techniques in detail. This provides the background necessary to introduce the main topic of the thesis, namely segmentation of medical images using the level set method.

2.1.1 Definition

A general definition of image segmentation is

“*Segmentation* in the image processing sense is the process of dividing or partitioning a digital image into a set of regions, where the regions correspond to objects of interest.”

Let us examine this definition in detail. “Segmentation” is a general word in the English language meaning to divide some whole into smaller parts, but in this definition segmentation is limited as describing a process of converting a *digital representation of an image* into a mathematical set. The set produced is defined as a *set of regions*, where each region is said to correspond with one object, which is described as *object of interest*. Let us further examine the three key elements, the digital representation of an image, the set of regions and the objects of interest.

To give more vivid examples of the concepts, we will refer to the example image shown in figure 2.1, which shows a busy street scene in rural Shanghai.

Digital representation of an image. Everybody intuitively understands what an image is. However, to be able to discuss images and operations on them in a more exact way, it is necessary to have a formal specification. We follow [51] and see images as mathematical functions. In general, the domain of the function is n -dimensional, and the domain and range of function values can be continuous or discrete. In case both the domain of the function and the range is discrete, the function is called *digital*. In this thesis, we only deal with such digital functions of images.

The most common input domain of a function I representing a digital image are two dimensional (2D) Cartesian coordinates (x, y) . The function value $I(x, y)$ represents a physical quantity measured. In the simplest case the quantity is an one dimensional one, such as brightness, temperature, distance or pressure. In some cases, the output range of the function I is multidimensional, for example with color images. The image function I for a color image maps to a coordinate in a color space. For the 2D case, one element of the picture is called *pixel*, short for *picture element*.

Higher dimension input domains than 2D also exist. For example, images with three dimensions (3D) can commonly be found in MR imaging, where a single element of the image is called *voxel*, for *volume element*. Time series of 3D images in turn could be represented as 4D images.

Summarizing, we define a digital image I formally as

$$I : \mathbb{N}^n \rightarrow \mathbb{N}^k,$$

where n is the dimensionality of the bounded domain, and k is the output dimensionality, where each dimension is made up by discrete levels.

For our example image in figure 2.1, the image is a 2D gray scale image of the brightness at a discrete resolution of 1024x768, with 256 discrete levels in the intensity range.

Set of regions. As result of the segmentation a set of regions is produced. Each region in turn is a set of image elements belonging to that set. The grouping of these elements into regions states a relationship between them: they are believed to belong to the same object. That is, the image elements in one region share a set of properties, they are said to be *similar*.

Between different adjacent regions there is *discontinuity*. Most often, all elements in each regions have to be connected with each other. This is one constraint which can be applied on each region, the *connectivity* constraint. Often, each region also has to fulfill a certain *regularity*, such as being smooth to some degree or have a fixed topology.

In our example figure 2.1 the set of regions depend on the objects we are interested in, which we discuss now.

Objects of interest. The object of interest is the entity present in the image that we are interested in for further analyzing. For example in a medical x-ray image we are interested in the shape of the bones, hence a segmentation algorithm might attempt to recover the shapes of bones from the image.

But the notion of an entity as a well defined element distinct from others is not always so clear. Consider a scanned image of a text document, with English characters organized in words and sentences. Instead of objects in the image, we have a whole hierarchy of objects. At a low level we have single elements, points, strokes and curves. Together these build single characters, which are in turn organized in words. Words are structured in sentences and paragraphs. The interest defines where in this hierarchy we draw the boundaries between objects and prescribes the *level of detail* at which the segmentation happens.

Regarding our example image in figure 2.1, we have no information about what a segmentation step should produce. But lets consider two plausible goals, (a) the segmentation of cars in the image and (b) the segmentation of the Chinese characters contained in the image.

For the first case, the segmentation of cars, the problem is tricky, as some cars are occluded by cars closer to the viewer. Also, the cars are of different shape, color and texture. Together,

it would quite difficult to build a segmentation algorithm targeting car shapes for static images. However, in case we have a sequence of images, we can use motion as powerful segmentation cue [18]. More generally, it is often easier to modify the input data modality to improve segmentation results rather than tuning the algorithm itself is.

The second case, segmenting the Chinese character also exposes interesting problems. As an abstract symbolic object, a character has a shape associated with it. But unlike cars, it is unclear how this shape is expressed in the image. Edges, changes in brightness, texture or other properties are enough for a human to properly recognize a character. For an automated segmentation we need a more strict definition of what constitutes a character. Unlike cars, one character may consist out of many non-connected shapes. Then, the symbolic high level object “character” relates to more than one connected region in the image. Again unlike cars, characters can appear at almost any *scale* in the image, up to the point that they are not identifiable anymore. A segmentation approach incorporating knowledge about characters hence must be able to apply this knowledge in a scale invariant way. More generally, the segmentation results capture shape information at a limited *level of detail*. Thus, even if a shape resulting from an object of interest is present in the image, if its level of detail inside the image is outside of the range captured by the segmentation algorithm it will not be properly segmented.

2.1.2 Properties of segmentation algorithms.

Following Suri et al. [53], we now give a list of desirable properties a medical image segmentation algorithm should possess. A good medical image segmentation algorithm has:

- Accurate segmentation results.

An accurate segmentation represents the original structure of interest well and reliable. There are two fundamental limitations to the accuracy of the segmentation, that is

- the quality of the input data limits the quality of the resulting segmentation, and
- the structure of interest must be well defined.

- Flexible topology.

The segmentation algorithm must be able to handle simple structures as well as complex structures with high curvature, such as the brain, or structures with holes.

- Minimal use of initial parameters.

For practical applications, the initial parameters to be determined manually should be minimal.

- Converging and stable.

The algorithm should converge to the final segmentation result. Convergence has two advantages, namely (a) a stopping criterion can be build easily for the segmentation process, and (b) the segmentation can be performed automatically because it is robust to different initialization parameters.

- Robust against local noise.

The quality of the segmentation should not be affected by local noise.



Figure 2.1: 2D gray scale intensity image of a busy street scene.

- Broadly applicable.

The algorithm should be applicable to a large number of image modalities, such as CT, MRI and ultrasound systems, as well as to 2D and 3D uni- and multimodal images.

- Reasonable performance.

The segmentation algorithm must be reasonably fast and run within acceptable space constraints.

- Flexibility.

The algorithm must be flexible for extension for two reasons. First, the technical imaging systems will change in the future, and by being flexible it might be possible to use extensively tested segmentation algorithms on new systems with minimal changes. Second, the algorithm may have to be adapted to incorporate extra information into the segmentation, such as model-driven knowledge about the structures to be segmented.

2.1.3 Applications of Medical Image Segmentation

The currently most dominant applications of reliable segmentation algorithms are 2D and 3D visualizations of the internal structures of the patients body, enhanced diagnostics and detailed planning of surgeries.

- *Volume Visualization.*

Through accurate segmentation and surface extraction algorithms, the two-dimensional image slice data created by MRI, MRA, CT and ultrasound imaging can be converted to surfaces embedded in a three dimensional volume. These surfaces and volumes can be displayed using *rendering techniques*. The clinical value of this visualizations is extensively studied by Sakas et al. in [44].

- *Enhanced diagnostics.*

Knowledge about the exact geometric shape of interna of the patients body allows enhanced diagnostics, such as detailed volume measurements. Malladi et al. use level set techniques in [30] to take exact measurements of the volume of bones and muscles from MRI images. Later, Sethian and Malladi extend this work in [47] to segmentation of the liver, the heart chambers, skull and the complex structure of the brain.

The use of three-dimensional ultrasound in prenatal diagnostics and fascinating visualization of the unborn baby's face is detailed in [44].

- *Planning of surgeries and treatments.*

Accurate segmentation of medical image data provides the basis for detailed three-dimensional planning of surgeries, virtual endoscopy and computer assisted radiation therapy planning. Actual examples and proposed virtualizations of important medical procedures are given in [44]. An introductory overview can be found in [2].

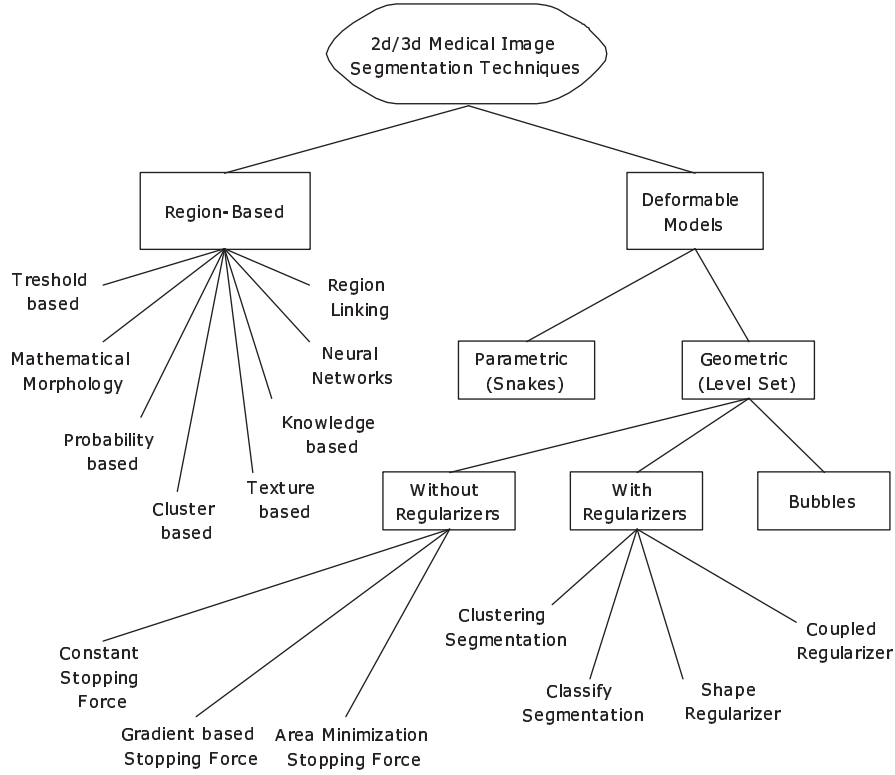


Figure 2.2: Medical image segmentation algorithm taxonomy tree following Suri et al. [53].

2.2 Medical Image Segmentation algorithms overview

An overview of segmentation algorithms used in medical image processing is shown in figure 2.2.

The main trees shown in figure 2.2 are region-based segmentation algorithms and segmentation algorithms based on deformable models.

Region based segmentation

Region based segmentation methods group pixels together based on some notion of similarity but do not have a knowledge of the resulting regions' shape. Below, we discuss in detail threshold methods, the watershed segmentation method based on mathematical morphology and region linking methods. Introductions to most of the region-based segmentation algorithms can be found in popular image processing textbooks, such as [18, 7, 15, 51].

Deformable model segmentation

The other large branch of segmentation algorithms deals with deformable models. Deformable models, of which level sets are an important subset, describe a shape in an image by its contour instead of the region it occupies. The contour is evolved to match the shape of the object to be segmented. One famous branch of deformable models used for segmentation are *Snakes*, described in section 2.6. The other major subbranch is based on level sets, which we describe in a dedicated chapter. The fundamental difference between the parametric and geometric

models is the kind of representation used for the curve or surface. Parametric models are explicit, while geometric models are defined implicitly as a property of a higher level function. The branch of level set methods is further subdivided into three classes, geometric deformable models without regularizers, with regularizers and bubbles. We now discuss them briefly and point out important recent trends in the field.

Geometric deformable models without regularizers. The models without regularizers are evolved based solely on the information of the underlying image. A special case is an evolution happening at a constant speed normal to the curve or surface¹. No information about the shape, such as curvature, is used to make the curve more regular.

Almost all the algorithms without regularization can most efficiently be implemented using the Fast Marching Method (FMM), which we explain in chapter 4.

Deformable models without regularizers are very popular; Droske et al. [12] for example use an adaptive sized grid to embed the level set function. As an example application they evaluate the performance by semi-automatically segmenting glioma in MRI images of the brain.

Baillard et al. [4] use level set methods to segment complex 3D structures from MRI and ultrasound images, with experimental data given for segmentation of the brain.

While the segmentation problem I deal with within this thesis is limited to two region segmentation, there have been efforts to extend Level Set Methods to more than two regions to be segmented. In [8], Chan and Vese did so and extended the two-phase segmentation using Level Set Methods to multiphase segmentation.

Geometric deformable models with regularizers. Deformable models with regularizers include all of the above models but allow for the definition of a *regularization term*. This term adds a shape dependent force to make the results of a segmentation “regular” by some criteria. A common criteria is smoothness, such as continuity in the first derivative and curvature. We discuss the regularization terms in more detail in section 5.3.3.

Models with regularizers are very popular; we give some examples here. Vemuri and Guo developed a hybrid model of parametrized “snake pedals” in [55] and later detailed it in [56]. Their model can be used in the level set framework to fit a level set function to a prior model curve or surface. A similar approach is taken by Leventon, Grimson and Faugeras in [27]. They interpret the signed distance level set functions of segmented training shapes as higher dimensional points and apply PCA² to identify the principal model shapes. They incorporate an estimating final shape term into the level set evolution equation itself and guide the evolution into the most likely shape form. Both approaches are categorized among the *shape regularizers* segmentation methods in figure 2.2.

Zeng et al. [14] extended the level set segmentation procedure to evolve two surfaces at the same time. The surfaces are coupled by a distance conserving term which acts as a force on both surfaces to always keep the same relative distance to each other. He uses this coupling to achieve excellent results in the segmentation of white and gray matter in MRI images of the brain. The surface coupling imposes a regularization force which counters extending the segmented shape in case the surfaces “disagree”, while having no opposing effect when they

¹This important subset will be examined in detail when we are concerned with reinitialization of the level set function in section 3.3.1.

²Principal Component Analysis, see [13].

agree. Zeng's approach is a prime example for the *coupled regularizer* class of segmentation algorithms.

An interesting approach to gradually refine the segmentation is given by Lin, Yu and Duncan in [28]. They operate on two-dimensional ultrasound images which are incrementally blurred within a Gaussian pyramid. The segmentation starts on a coarse scale and the results of each scale are used both to initialize and to bound the segmentation process on finer scales. This bounding is a regularization force on the finer scale segmentation. They validate their approach using real ultrasound data.

Another effort to improve level set segmentation results by merging a-priori knowledge or regional statistics into the speed function is Ho et al. [22], who replace the propagation term with a force based on regional statistics and let adjacent regions compete for a common boundary. They demonstrate improved results for brain tumor segmentation in MRI. Suri used fuzzy classifications of regions in [52] to build a speed function incorporating shape, region, edge and curvature information and applied the resulting model successfully to brain segmentation.

Bubbles. Bubbles are simple object descriptions that are initialized globally and partially randomly on an image. These simple objects are then evolved nearly identical to the classical levelset evolution, using the reaction-diffusion equation

$$\frac{\partial C}{\partial t} = S(x, y)(\beta_0 - \beta_1 \kappa) \vec{N}, \quad (1)$$

where C is a bubble, $S(x, y)$ a stop function based on the image and β_0, β_1 are reaction- and diffusion term constants respectively. Numerically, the equation can be discretized using the levelset techniques outlines in chapter 3. The stop function $S(x, y)$ guides the growth based on local image properties, such as gradient. Its meaning is identical to the level set segmentation speed functions, and commonly $S(x, y) = \frac{1}{1+|\nabla G_\sigma * I(x, y)|^m}$ is used. During the evolution the bubbles are allowed to change topology, to merge, split or vanish.

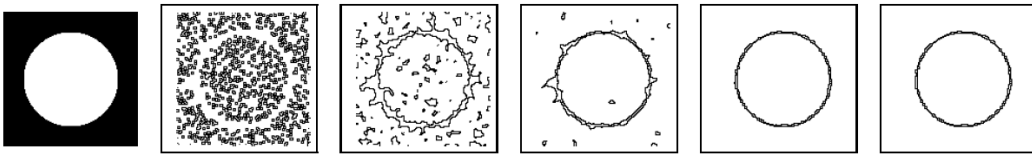


Figure 2.3: Example bubble evolution. The original image on the left is to be segmented by bubbles. First bubbles are randomly initialized in homogeneous areas of the image (second image from left), which successively are subject to evolution under equation 1. Images taken from [54].

Initially, the bubbles are randomly placed on the image where the image is uniform homogeneous with respect to the stop function S . Using the above function, this would place the bubbles at regions with a small gradient. This global initialization allows the bubble method to hypothesize about object components, while the following reaction-diffusion process validates, modifies or annihilates them. This process is shown in figure 2.3.

While closely related to the level set method from an implementation point of view, Tek and Kimia [54] also provide a strong theoretical and mathematical justification for their

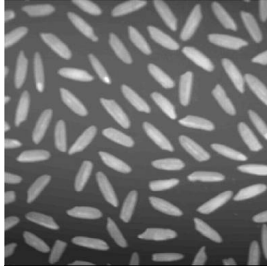


Figure 2.4: Gray scale image of objects to be segmented.

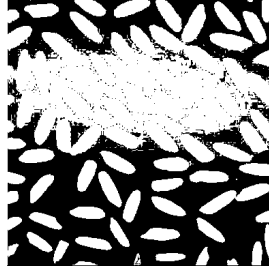


Figure 2.5: Image 2.4 thresholded at $T = 94$.

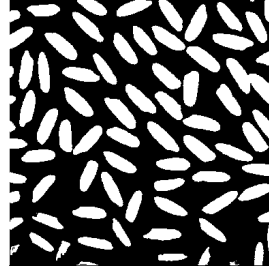


Figure 2.6: Image 2.4 thresholded at $T = 109$.



Figure 2.7: Image 2.4 thresholded at $T = 167$.

Bubble model by building a connection to shock-based description of shapes, reducing shape in a hierarchical representation to four fundamental type of shocks [26].

Summarizing, bubbles can solve some of the important problems occurring in medical image segmentation, such as the segmentation initialization. Additionally, the use of the reaction- and diffusion term constants can be used to balance result towards the expected shapes.

2.3 Thresholding based segmentation methods

2.3.1 Basic thresholding

Basic thresholding has been popular since the beginnings of digital image processing [17, 18]. The idea is to create a region membership function based on the intensity and then to apply this function to every single pixel in the input image to obtain a region membership map. This membership map groups the pixels into regions, which is the segmentation result.

More formally, for a two-region case a function $f(I)$ is used

$$f(I) = \begin{cases} 0 & \text{if } I \leq T \\ 1 & \text{if } I > T \end{cases},$$

where I is the intensity value, and T the *threshold value*. For more than two regions, the function similarly divides the input intensity range into intervals, assigning one set index for each.

Consider the gray scale image shown in figure 2.4. The histogram, the relative count of the different intensity values is shown in figure 2.8. In figures 2.5, 2.6 and 2.7 the resulting thresholded sets are shown for the values $T = 94$, $T = 109$ and $T = 167$ respectively. If our interest lies in separating the small objects from the background, clearly $T = 109$ gives the best result. For $T = 94$ part of the background is included in the positive set, for $T = 167$ the positive set does not include parts of the objects in the lower right corner of the image. Generalizing from this example, we now analyze the properties of the basic thresholding procedure.

While for simple cases the basic thresholding method is sufficient, the results achievable are limited by four factors:

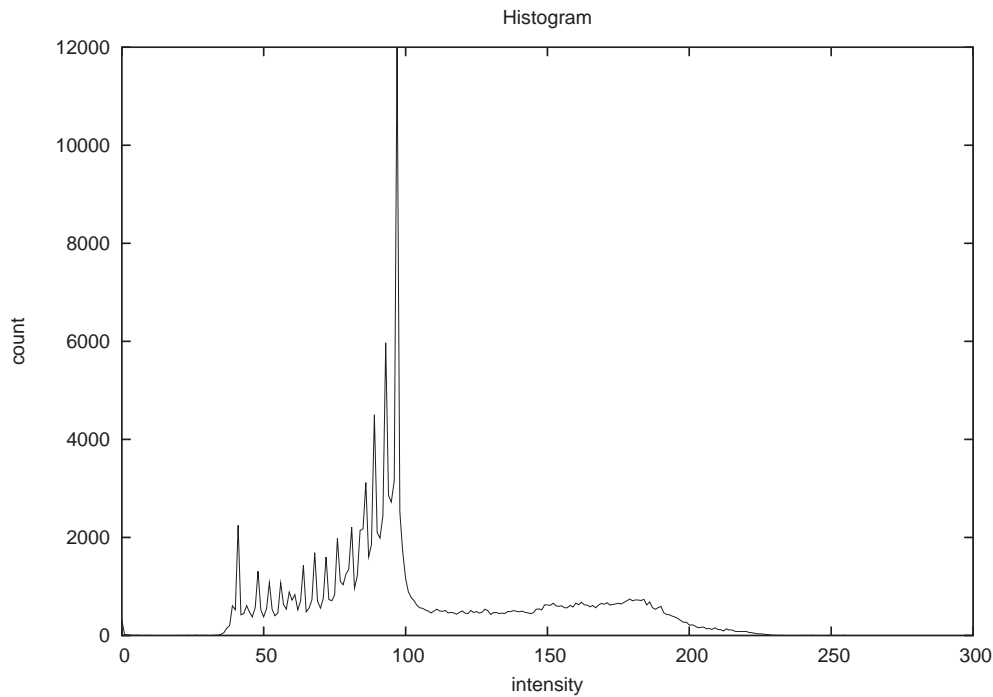


Figure 2.8: The intensity histogram of the gray scale image shown in figure 2.4.

1. Only a single pixel value is considered at a time.

The region classification decision is made based only on one pixel's intensity value, without considering its neighborhood or the overall image structure, discarding all spatial information.

We will discuss *local thresholding* below, which introduces information about the neighborhood into the classification decision.

2. Thresholding uniform over the entire image.

The image may be uniform in structure, or it may be not. Basic thresholding can deal well with the uniform case, where the intensity of one region does not change. When the intensity varies, the segmentation performance decreases. *Local thresholding* leads to an improvement in this case.

3. Threshold value T strongly dependent on the image.

The only free parameter T defines the resulting segmented regions. To choose the “right” value of T , that is, the value producing the minimal segmentation error, is generally impossible for all images. It is strongly dependent on the input image, which makes choosing a good default value difficult. The *adaptive thresholding* methods we discuss below try to obtain a good threshold value T automatically.

4. No geometric interpretation.

No geometric interpretation is done to obtain the classification decision; the resulting segmented regions have no regular geometric properties, such as “being smooth”. In

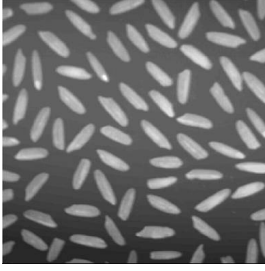


Figure 2.9: Gray scale image of objects to be segmented.



Figure 2.10: OTSU algorithm, $T = 125$.



Figure 2.11: Entropy method, $T = 121$.



Figure 2.12: Isodata algorithm, $T = 124$.

general, threshold based segmentation algorithms can not overcome this limitation, as it is inherent in the approach of dealing with a small part of picture at a time.

2.3.2 Adaptive thresholding

In this section we give a brief introduction to three common generic threshold algorithms, the OTSU algorithm, the Entropy method and the Isodata algorithm. For many applications there exist modified versions of the algorithms presented here or novel schemes producing better results. The algorithms given here are still popular as preprocessing-step on images, but have fallen out of use for segmentation due to the invention of more powerful methods.

OTSU algorithm

The OTSU algorithm, named after its creator Nobuyuki Otsu, and published in [41] is a well known algorithm to automatically derive a “good” threshold value to use with binary thresholding.

We first define

$$p(i) = \frac{n_i}{N},$$

where n_i is the number of pixels which have an intensity value of i , and N is the total number of pixels in the image. Next the frequencies ω_0 and ω_1 for each class are defined as

$$\omega_0 = \sum_{i=0}^T p(i), \quad \omega_1 = \sum_{i=T+1}^{255} p(i).$$

The mean values are obtained using

$$\mu_0 = \sum_{i=0}^T \frac{ip(i)}{\omega_0}, \quad \mu_1 = \sum_{i=T+1}^{255} \frac{ip(i)}{\omega_1},$$

and additionally the total mean is calculated as

$$\mu_T = \sum_{i=0}^{255} ip(i).$$

The OTSU algorithm chooses the threshold value T which maximizes the term

$$\eta = \omega_0 \cdot (\mu_0 - \mu_T)^2 + \omega_1 \cdot (\mu_1 - \mu_T)^2.$$

For the example image, the OTSU algorithm chooses $T = 125$, the result is shown in figure 2.10.

Entropy algorithm

The entropy threshold value selection method introduced by Wong and Sahoo in [57] works by measuring the information theory entropy values for the classification sets. These entropy measures, H_0 and H_1 are defined as

$$H_0(t) = - \sum_{i=0}^t p_i(0) \log(p_i(0)) \quad H_1(t) = - \sum_{i=t+1}^{255} p_i(1) \log(p_i(1))$$

where $p_i(k) = \frac{n_i}{N_k}$, $k \in \{0, 1\}$ denotes the quotient of the number of pixels n_i that have the intensity value i , divided by the number of total pixels N_k in the class 0 or 1.

Then, the threshold value is selected by choosing the value t that maximizes $H(t)$, defined as

$$H(t) = H_0(t) + H_1(t).$$

For the example image, the entropy method chosen threshold value is $t = 121$. The result of the thresholding is shown in figure 2.11.

Isodata algorithm

The so called Isodata algorithm introduced by Ridler and Calvard in [43]³ iteratively finds a threshold by taking the sample mean value of the gray values associated with each class.

Algorithm *IsodataThreshold*

Input: A histogram $h[0 \dots n]$ of an image.

Output: A threshold value T to use with binary thresholding on the image.

1. $T \leftarrow \frac{n}{2}$
2. **repeat**
3. $T_l \leftarrow T$
4. $R_0 = h[0 \dots T]$
5. $R_1 = h[T + 1 \dots n]$
6. $\mu_0 \leftarrow \text{mean intensity of } R_0$
7. $\mu_1 \leftarrow \text{mean intensity of } R_1$
8. $T \leftarrow \frac{\mu_0 + \mu_1}{2}$
9. **until** $T_l = T$
10. **return** T

For the example image, the isodata algorithm selects a threshold value of $T = 124$, the result is shown in figure 2.12.

³With additional good comments available in [29].

2.3.3 Local thresholding

The above thresholding algorithms can be improved by adapting the threshold to local image regions. That is, instead of only one threshold value T for the entire image, different values are used for different parts of the image. This makes it possible to deal with changes in illumination. Details of different local thresholding methods are given in [18].

2.4 Region based segmentation methods

In region based segmentation algorithms, the image is interpreted as a region which is the sum of disjoint, connected partitions. Formally, the complete image region R is defined as

$$R = \bigcup_{i=1}^n R_i, \quad R_j \cap R_i = \emptyset \quad \text{for all } i, j, i \neq j,$$

where R_i is an individual connected subregion.

Two popular region based segmentation methods using this definition are *region growing* and *region splitting and merging*, which we discuss briefly now.

2.4.1 Region growing

Region growing produces the set of regions R_i to satisfy the above definitions using an iterative algorithm. Initially there are *seed regions*, of which regions with just one element – *seed pixels* – are a common special case. In each iteration the neighbors of each region which do not belong to any region yet are checked if they satisfy a *similarity criteria*. The end of the iteration process is controlled by a *stopping rule*.

Similarity criteria. The similarity criteria decides about whether to join a neighboring element of a region into the region or not. It is strongly dependent on the segmentation problem itself and the image modality.

Usually the similarity criterion is designed to base its decision on the absolute intensity⁴, the relative intensity difference and the gradient, texture, etc.

Stopping rule. The similarity criteria decides when a neighboring pixel shall be merged into the region, but still is entirely local in its nature. That is, the global properties of each region, such as size or contour length are not considered. By allowing an additional stopping rule to be based on such region-global properties the power of the region growth method is improved.

Comparing region growing to basic thresholding. The higher segmentation performance of region growing segmentation stems from its additional use of *connectivity*. In the basic thresholding methods we discarded all spatial information, including the inter-relationships of the picture elements. Connectivity is a simple, but one of the most important relationships between picture elements in regard to segmenting single objects.

We will see in the following sections how more and more spatial information is incorporated into the segmentation process, improving the results.

⁴Or for multicolor images, a region in a color space.

ridge that separates the valleys is overflowed by the rising water. Instead of allowing the two basins to merge, the watershed segmentation algorithm constructs an artificial dam that separates the valleys. When the whole landscape is covered by the water, the segmentation stops.

Algorithm. We now reconsider the intuitive idea outlined above in a more formal and algorithmic approach. One implementation of the watershed segmentation is shown below in the *WatershedSegmentation* algorithm.

Algorithm *WatershedSegmentation*

Input: The intensity image I , a dilation mask M .

Output: The set of segmented regions R , the built set of dams dam .

```

1.   $min \leftarrow$  minimum intensity in  $I$ 
2.   $max \leftarrow$  maximum intensity in  $I$ 
3.   $R \leftarrow \emptyset$ 
4.   $dam \leftarrow \emptyset$ 
5.  for  $height = min$  to  $max$ 
6.       $New \leftarrow \{(x, y) | (x, y) \in (thresh(I, height) \setminus thresh(I, height - 1))\}$ 
7.       $NewC \leftarrow$  set of connected components in  $New$ 
8.       $R \leftarrow R \cup (NewC \setminus \{e \in NewC | \{(x, y) \in e | (x, y) \in r \text{ for any } r \in R\} \neq \emptyset\})$ 
9.       $New \leftarrow New \setminus \{(x, y) \in r, r \in R\}$ 
10.     while  $New \neq \emptyset$ 
11.          $D \leftarrow ()$ 
12.          $P \leftarrow \emptyset$ 
13.         for region  $r_k \in R$ 
14.              $D_k \leftarrow Dilate(r_k, M) \cap New$ 
15.              $dam \leftarrow dam \cup (P \cap D_k)$ 
16.              $P \leftarrow P \cup D_k$ 
17.          $New \leftarrow New \setminus P$ 
18.         for region  $r_k \in R$ 
19.              $r_k \leftarrow r_k \cup (D_k \cap dam)$ 
20. return  $R, dam$ 

```

The function $thresh(I, height)$ is defined as

$$thresh(I, height) = \{(x, y) \in I | I(x, y) \leq height\}.$$

The function $Dilate(r, M)$ is the standard binary mathematical morphology dilation operation for all elements in r using the dilation mask M .

The implementation is straightforward. The main loop in line 5 increases the water level in each iteration. The newly flooded elements are identified in line 6, then grouped into connected components. Those components that have no interconnection to already known regions are added to the set of regions in line 8, as they represent distinct valleys. The remaining pixels which are connected with known regions are kept in line 9 while the pixels belonging to the new regions are discarded. The following iterating loop in line 10 deals with only these remaining pixels.

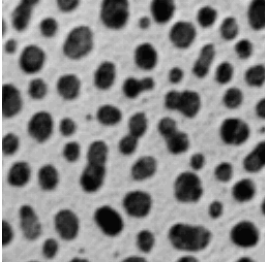


Figure 2.14: Blobs to be segmented.

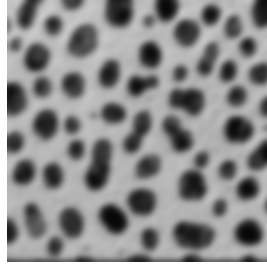
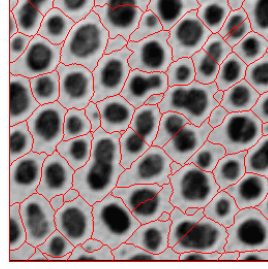
Figure 2.15: Gaussian smoothed image 2.14, $\sigma = 3.0$.

Figure 2.16: Segmented watershed/divide lines.

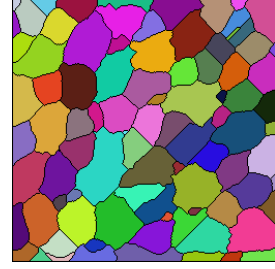


Figure 2.17: Watershed catchment basins.

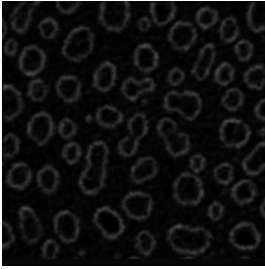


Figure 2.18: Normalized gradient magnitude image of image 2.14.

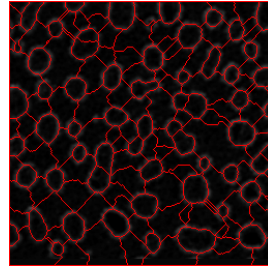
Figure 2.19: Gaussian smoothed image 2.18, $\sigma = 3.0$.

Figure 2.20: Segmented watershed/divide lines.

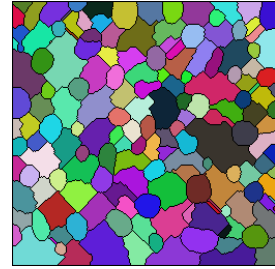


Figure 2.21: Watershed catchment basins.

In the loop a successive dilation is performed using the supplied dilation mask D^5 . In each dilation iteration pixels belonging to more than one region are identified and a dam is created at their place (line 15).

The dams, also known as *watershed lines* form connected pathes, separating the segmented regions.

In most cases, the watershed segmentation is not done on the original image, but on the gradient magnitude of the Gaussian smoothed image, $|\nabla G_\sigma * I|$. The smoothing using the Gaussian reduces noise and smoothes the resulting landscape, improving the robustness of the segmentation. The larger the smoothing parameter σ , the smaller the number of regions found⁶. The gradient image is often used, because the resulting dams are located at large gradient magnitudes, which correspond to large intensity changes in the original image. Such large intensity changes happen at the edges of the objects to be segmented if the object intensity varies from the background intensity.

Example. Lets consider the example⁷ shown in figures 2.14 to 2.17.

The original image 2.14 shows a number of different sized dark objects on a white background. The segmentation should identify regions, each containing only one such object. In

⁵Normally a 3x3 cross or a 3x3 square mask is used for the dilation.

⁶Commonly values $\sigma = 1.0$ to $\sigma = 5.0$ are used, but this can vary depending on the relative resolution of the image, the noise present in it and the regions of interest to be segmented.

⁷Images provided by the Biomedical Imaging Group, EPFL, <http://bigwww.epfl.ch/sage/soft/watershed/>.

this case, the segmentation is directly applied on the smoothed image $G_\sigma * I$, not on the gradient image. A parameter $\sigma = 3.0$ produces the smoothed input image 2.15 to the segmentation algorithm. The dams are shown overlayed on the image in figure 2.16 and the identified regions are shown in figure 2.17.

The same segmentation procedure is applied in figure 2.18 to 2.21, but as input picture the normalized gradient magnitude image of the image shown in figure 2.14 is used. The resulting watershed lines lie on places of strong intensity changes, in this case on the boundary of the blobs. In the final result shown in figures 2.20 and 2.21 every blob has its own region, but oversegmentation is apparent.

Problems and improvements. A general problem with the watershed segmentation is oversegmentation, leading to regions that do not correspond to actual objects of interest. Some of the oversegmentation can be suppressed using a Gaussian smoothing step or by introducing additional constraints, such as a minimal region size or catchment basin depth. However, the underlying cause for the oversegmentation is the global nature of the watershed algorithm, which adds new regions whenever a new local minimum is found, disregarding any information about the segmentation history and the other regions present. Such local minima exist at noisy parts and local irregularities of the image. The Gaussian smoothing reduces them to some extent but also deteriorates the segmentation performance by reducing the number of regions that can be found.

The segmentation results can be improved by incorporating previous knowledge about the structure of the image into the process. A straightforward way to do this is the use of markers which explicitly mark known region seeds and regions not belonging to objects of interest. Often a natural way to obtain such seed regions exists for a given segmentation problem. For example, in the image shown in figure 2.14 a simple thresholding step followed by binary morphological erosion could have been used to obtain seed markers for the blobs. The use of markers is detailed in [18].

Since its invention in [6], a large number of modifications have been proposed to the watershed segmentation algorithm which are too numerous to list here. The main improvements can be grouped into four groups: runtime performance improvements, adding preprocessing and constraints, incorporating additional problem-specific knowledge and the combination with other segmentation methods. A good overview is given in [36].

2.6 Snakes - Parametric active contour segmentation methods

Snakes were the first deformable model used for segmentation. Introduced by Kass et. al [24] in 1987, a snake is a parametrized curve $C(s) = (x(s), y(s))$, which has an associated energy term $E(C)$, defined originally by Kass as

$$E(C) = \int_0^1 [E_{int}(C(s)) + E_{image}(C(s)) + E_{con}(C(s))] ds. \quad (2)$$

A segmentation using snakes minimizes the energy term $E(C)$ for C . In a successful segmentation, the resulting energy-minimized snake C correspond to the boundary of the object of interest in the image. The energy term in equation 2 itself is composed from three energies, namely

1. E_{int} , the internal force controlling the snakes rigidity and elasticity,

2. E_{image} , the image dependent force attracting the snake to the object boundaries, and
3. E_{con} , the external force (also called *constraint energy*) controlling the snake by means of user input,

acting on the snake C . A common form of equation 2 is given in [39] as

$$E_1(C) = \alpha \int_0^1 |C'(s)|^2 ds \quad (3)$$

$$+ \beta \int_0^1 |C''(s)|^2 ds \quad (4)$$

$$- \gamma \int_0^1 |\nabla u_0(C(s))|^2 ds. \quad (5)$$

The function of E_{int} is taken by the squares of the first and second derivatives $C'(s)$ and $C''(s)$, terms 3 and 4. The constants α and β control how much they influence the snake. The E_{image} energy is found in term 5, where ∇u_0 is the image gradient⁸. The user controlled energy E_{con} dropped out of equation 2.

2.6.1 Parametrization methods

To implement the snake model described above, the snake $C(s)$ has to be parametrized. Commonly polygonal approximations are used, of which the simplest one are piecewise linear approximations. A more powerful and commonly used parametrization is based on Bézier-Splines. The snake is parametrized using the B-spline expression

$$\forall s \in [0; 1], \quad C(s) = \sum_{i=0}^{i=n-1} P_i \alpha_i(s), \quad (6)$$

where $P_i \in \mathbb{R}^2$ are the n control points and α_i is the spline basis function. To obtain a discrete polygonal approximation of the contour from the above parametrization the Oslo algorithm [35] is used. The evolution of the curve happens on this polygonal approximation by first moving the control points and then reconstructing a new B-Spline curve from the new control points. This evolution step is detailed in section 3.1.1.

An overview of the issues involved in approximating snakes using B-Splines is given by Cottet et al. in [11]⁹.

2.6.2 Problems of the original Snake model

We give a brief list of the common problems in the original snake approach. The problems inherent to the representation of the contour will be examined further in section 3.1.1.

The original snake approach has the following limitations:

⁸Normally, u_0 is a presmoothed version of the original image, so that $u_0(x, y) = G_\sigma * I(x, y)$ with G_σ being a Gaussian filter.

⁹A more general introduction to B-Splines and other interpolation methods is given by Heath in [21].

- Parametrization.

Without prior knowledge about the object of interest in the image, it is unclear how to choose the number of parametrized segments. A simple shape is better captured by few segments, whereas a convoluted shape requires a more complex parametrization.

- Initialization.

Prior information about the location and size of the object of interest in the image is necessary to properly initialize the snake. Otherwise it might fail to capture the shape of the object.

- Merges and splits.

In case the contour merges with itself or splits due to regularization constraints in E_{int} , the original snake method fails to adapt the topology of the snake¹⁰.

- Loops.

During evolution loops might appear when two adjacent parametric segments cross over each other. These loops are problematic as they make the numerical evolution unstable, as examined in section 3.1.1.

- Use of image information only at the contour.

The image dependent part of the evolution is guided only by the local image information around the contour. Structure away from the contour is ignored.

- Non-adaptiveness.

The choice of the free parameters in equations such as 3 has to be carried out manually by the user.

2.6.3 Proposed solutions

There exist many variations of the snake segmentation method, which address one or more of the above problems. We briefly reference the most important ones here.

The problem of being unable to handle topological changes has been addressed by McInerney and Terzopoulos in [33] and [34] for both the 2D and 3D cases. They propose *T-Snakes*, topology adaptive snakes. By discretizing the domain using *affine cell image decomposition* (ACID) into convex polygons topological change is detected and handled efficiently. A drawback of the approach is the introduction of the unintuitive extra discretization of the image.

The problem of looping is commonly addressed using a *delooping* step after each evolution time step. In this delooping step, loops are identified, the intersection points located and a new snake is constructed omitting the loop [23]. For 3D this becomes considerably more difficult, as surface intersections have to be computed.

In [19], Nixon and Gunn address the initialization and adaptiveness problems. Instead of using a single contour, they use two opposing snakes, one evolving from the inside of a hypothesized object, one from its outside. This improves the robustness, but works only in

¹⁰As long as only the first and second order derivatives are used the contour cannot split for the two-dimensional case. It has been shown that a curve evolving under its curvature will collapse to itself without splitting.

cases where some location and shape information is already known about the object to be segmented.

Xu et. al introduce two extra forces in [58] to make the Snake more robust to initial parameters and location. They achieve this by countering variations in normal forces arising from local contour shape, and add a normal force that is identical for all points on the contour, to adjust the growth or shrinking process. This extension provides additional robustness but at the cost of more terms to the original snake equation 2.

2.6.4 Conclusion

Snakes are powerful, efficient and well researched tools to recover shape information from medical images. The problems of segmentation using Snakes are well known and understood. While there are proposed solutions, there is still a lack of a simple, elegant and integrated snake model solving all of them. An important common disadvantage of all proposed solutions is the cost of further terms, some of which are unintuitive, ad-hoc and difficult to combine with other approaches.

Snakes will continue to be used successfully in medical image segmentation. The theoretically more elegant and flexible contour model used in the level set methods is an alternative that provides significant advantages while still allowing most of the successful snake methods to be implemented.

Chapter 3

The Level Set Method

In this chapter we introduce the level set method, which deals with the representation and evolution of curves and surfaces, generally referred to as *interfaces*. By first explaining traditional methods we give the background necessary to understand the workings and advantages of the Level Set Method. Based on this understanding we give a practical framework to use the level set method in and give advice on how to implement it.

The Level Set Method is relevant to this thesis as it is the main ingredient in many modern image segmentation algorithms. In section 5.3 we introduce the segmentation based on level sets. What follows is the theoretical basis of this thesis's work.

3.1 Interface evolution

In the most general sense an *interface* is a boundary between two entities. In the course of this discussion an interface is a geometric description of a closed contour, such as a curve, surface or hypersurface, that separates the space in two domains, the *inside* and the *outside domain*. Then, *interface evolution* is the process of changing the interface according to evolution laws and constraints.

Interface representation and evolution has some obvious applications in the simulation of physical phenomena, such as in Computational Fluid Dynamics (CFD) and Computer Graphics (CG). In many other fields however, the notion of an interfaces is not so natural. Even so, in these domains there is a surprisingly large number of successful applications of interface methods, by reinterpretation of the original problem so the solution to the new problem can be obtained indirectly from the interface. This is the case for Computer Vision (CV) and Digital Image Processing, where interface methods have been used successfully.

The problem of interface representation and evolution occurs in many places and simulating and solving the evolution of interfaces has been a long time interest to scientists. As such many techniques have been developed, and we now discuss two traditional techniques, which – once dominant – have been largely replaced by the Level Set Method in recent years. The first technique is commonly referred to as *marker method* and describes the interface in what first seems like a very plausible way. It turned out that this method, while successful in many regards, had problems inherent in the approach, and the second method, the *volume-of-fluid method* was aimed to solve some of them. By pinpointing the problems of each of this methods we learn to appreciate the advantages of the Level Set Methods.

3.1.1 Traditional technique: markers

A popular and well documented method is to discretize the interface into a finite number of elements. For example, a curve in two dimensional space can be approximated arbitrary well using line segments. For surfaces in three dimensional space, triangles can be used. The representation and evolution of the interface using discretized elements to represent the interface itself is known under a number of names, such as marker particle technique, string methods and nodal methods [46].

Lets consider a curve in two dimensional space as shown in figure 3.1. As approximation, we use an ordered list of points $X = \{x_0, \dots, x_n\}$ which are connected circularly by $n + 1$ line segments. To have an equal distribution of accuracy, the points should be placed equidistant on the interface.

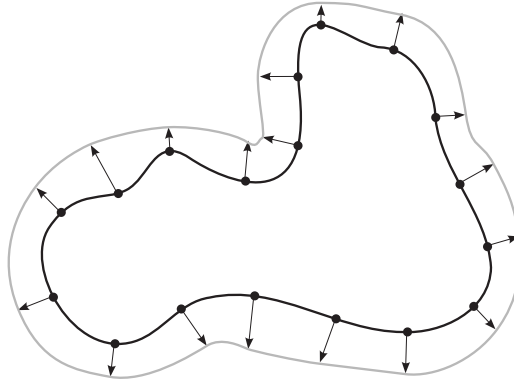


Figure 3.1: Example curve represented by markers.

We now describe how to evolve this curve over time given a speed function F . To evolve the curve, the time is discretized into fixed steps Δt . Then, for each point x in the set of points, a movement vector is calculated using the speed function and the point is moved by this vector. The moved interface is simply the collection of moved points. In this process, there are two important aspects, the movement vector calculation and a known speed function, which we discuss now.

Movement vector We assume the movement always happen in the normal direction of the curve and the speed function returns the signed speed in normal direction. The normal of the curve can be given as

$$\vec{n} = \frac{(y_s, -x_s)}{(x_s^2 + y_s^2)^{\frac{1}{2}}}$$

where x_s and y_s are the spatial derivatives $\frac{dx}{ds}$, $\frac{dy}{ds}$ in x and y directions at the given point (x_i, y_i) . Using a finite central difference approximation for a spatial step size Δs between the individual points one can derive

$$\frac{dx_i}{ds} \approx \frac{x_{i+1} - x_{i-1}}{2\Delta s}, \quad \frac{dy_i}{ds} \approx \frac{y_{i+1} - y_{i-1}}{2\Delta s}$$

For details the reader can refer to [46]. Using the normal vector we can construct the derivative over time for the point positions as the product of the normal relative speed function F with

the x and y component of the normal vector \vec{n} as follows

$$(x_t, y_t) = \left(F \cdot \left(\frac{y_s}{(x_s^2 + y_s^2)^{\frac{1}{2}}} \right), F \cdot \left(\frac{-x_s}{(x_s^2 + y_s^2)^{\frac{1}{2}}} \right) \right).$$

Speed function The speed function F gives the speed in normal direction for a point (x_k, y_k) . While the speed function can vary for each time step and point, often a curvature-dependent term is used to smooth the curve during evolution. In section 3.2.3 we explain the concept of curvature in general. In the marker method, the curvature is approximated using finite differences by replacing the second order spatial derivatives x_{ss} and y_{ss} in the general curvature definition

$$\kappa = \frac{y_{ss}x_s - x_{ss}y_s}{(x_s^2 + y_s^2)^{\frac{3}{2}}}$$

using the finite approximations

$$\frac{d^2 x_i}{ds^2} \approx \frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta s^2}, \quad \frac{d^2 y_i}{ds^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta s^2}.$$

Evolution equation Combining the above results, we can evolve every point position (x_i^k, y_i^k) at time step k to the new position (x_i^{k+1}, y_i^{k+1}) at time step $k + 1$ using

$$(x_i^{k+1}, y_i^{k+1}) = (x_i^k, y_i^k) + \Delta t F \vec{n}_i.$$

The movement of every individual marker in normal direction and the resulting curve is shown in figure 3.1.

Disadvantages of the marker method While the marker method has been used successfully in many fields, the underlying idea has some disadvantages that are not easy to cure.

1. Unstable.

The approach is unstable because small errors in the markers position are amplified during evolution. Because the calculation of the derivatives are based on the neighboring markers positions, small errors in the positions lead to errors in the derivatives. The derivatives are in turn used to calculate the marker velocities, which move the markers, further introducing errors in their positions. However, the amplification can be limited to ensure stability by using very small time steps Δt , which reduces the performance.

2. Loss of equidistancy.

Even when the evolution is accurate and the markers have originally been positioned in equal distances Δs on the interface, this property is not conserved. This leads to errors in the approximated derivatives.

3. Problems with singularities.

Consider a speed function that does not use the curvature to obtain a smooth curve. The evolution of a V-shaped curve using the speed function $F = 1$ is shown in figure 3.2. While the original curve at $t = 0$ is smooth, a sharp corner develops at the bottom of

the V, which finally leads to two parts of the curve crossing each other, where the crossed part is characteristically named “swallowtail”. Using the marker method there is no solution to this problem, but a workaround is known and applied in practice by removing the swallowtail manually after each iteration. This can be complex in three dimensions [46].

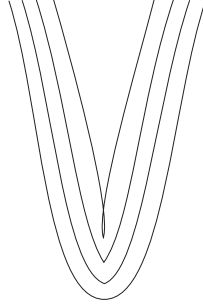


Figure 3.2: Marker method swallow-tailing the curve for $F = 1$.

4. Problems to handle topological change.

Related to the swallowtail problem is the problem of topological change of the front, that is, what happens when the front merges or splits. Using the marker method such a change has to be detected and the front has to be reconstructed much in the way the swallowtail is removed.

We have listed a number of disadvantages of the marker technique. Nevertheless the technique is still used in practice because it is well understood and able to model local front behavior well. Globally it is limited by its inability to naturally cope with singularities and topological change as well as by its instability. These limitations are inherent in the marker approach and even while there exist workarounds for each of the given problems, the method is fundamentally flawed in these respects.

3.1.2 Traditional technique: Volume-of-fluid

Another approach to represent an interface is the so called *Volume-of-fluid* method. The basic idea is a simple one: the computational domain is divided in a finite number of identical sized cells. Each cell is assigned a number between zero and 1.0, representing the relative amount of “fluid” in the cell, where zero means the cell is totally empty, and 1.0 means the cell is totally filled. An example curve is shown in figure 3.3.

While this representation is less accurate than the marker based one, it is the first step away from an explicit representation of the interface to an implicit one. Its main advantages over the marker method are:

- Topology can change.

The topology of the curve or surface can change without affecting the workings of the method or its representation of the curve.

- There are *entropy condition*-satisfying evolution methods.

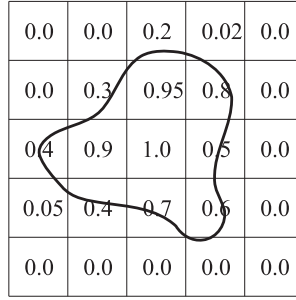


Figure 3.3: A curve represented as area in the Volume-of-fluid method.

In case the curve or area develops a singularity, such as a sharp corner, with the volume-of-fluid method it is still possible to obtain the correct solution, which is said to be a *weak solution*, satisfying the *entropy condition*.

- Area conservation.

During evolution it is possible to conserve the exact amount of area or volume occupied by the original curve.

The ideas behind the volume-of-fluid method ultimately lead to the development of the level set method. However, the volume-of-fluid method itself has disadvantages that are not easy to work around. Sethian [46] lists four: (a) the amount of resources to accurately represent the interface is high due to the inaccuracy of the volume-fraction approximation, (b) the evolution results are often dependent on the underlying orientation of the grid, (c) calculation of geometric properties such as the normal vector and curvature are inaccurate and (d) the volume-of-fluid method does not extend well to higher dimensions.

3.2 Level Set Method

Armed with an understanding of two traditional methods for interface representation and evolution, we can now examine the Level Set method in detail. The fundamental idea is similar to the Volume-of-fluid method, in that the interface is represented implicitly, but a different representation is chosen. We first examine this representation of the interface, and then specify important properties. The level set method was first introduced by Osher and Sethian in [40].

3.2.1 Implicit representation of an interface

A closed curve or surface can be represented implicitly by a function defined on a domain Ω . Consider the curve shown in figure 3.4. The domain Ω is two dimensional and bounded by the drawn box, so it completely contains the curve. The curve - which is one dimensional - divides Ω into two disjunctive sets Ω^+ and Ω^- so that $\Omega = \Omega^+ \cup \Omega^-$ and all elements in Ω that are located on the outside of the curve belong to Ω^+ and all elements on the curve or inside of it are in Ω^- .

Usually the domain is addressed using Cartesian coordinates and an *embedding function*

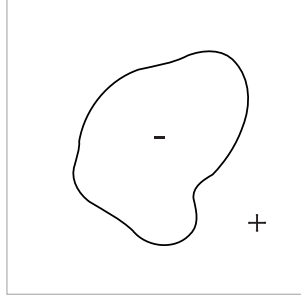


Figure 3.4: A curve represented implicitly by a function.

ϕ defined so that for each coordinate $c = (c_x, c_y)$:

$$\phi(c) = \begin{cases} \leq 0 & c \in \Omega^- \\ > 0 & c \in \Omega^+ \end{cases}$$

Not considering how to evolve the interface, this representation alone has some advantages:

1. Union, intersection and difference operators defined.

The geometric interfaces embedded into an implicit representation can be combined easily using the union, intersection and complement operators. The operators have a simple interpretation for the interface-representing functions, namely *min* (union), *max* (intersection) and negation for the complement. The resulting function is again an implicit function, however the resulting function may not conserve important properties of the original embedding functions.

2. Generalizes to higher dimensions.

The concept of embedding an interface in a function ϕ of a dimensionality one higher than the interface is applicable in any dimension.

There is an infinite number of functions that can be used to embed an interface. However, one function is particularly well suited for embedding an interface, the *signed distance function*. We now explain its characteristics in detail by giving an example and list reasons why it is a good choice.

3.2.2 The signed distance function

The *signed distance function* ϕ_{sd} for a two dimensional computational domain capturing a closed one dimensional curve is defined as follows:

$$\phi_{sd}(x, y) = \begin{cases} \left| \begin{pmatrix} x \\ y \end{pmatrix} - P(x, y) \right| & \text{if } (x, y) \text{ is on the interface or outside of it} \\ -\left| \begin{pmatrix} x \\ y \end{pmatrix} - P(x, y) \right| & \text{otherwise} \end{cases}$$

where $P(x, y)$ is one of the nearest point from (x, y) on the interface. Intuitively $\phi_{sd}(x, y)$ always tells the distance to the nearest point on the interface, but for points located inside of the curve the distance has a negative sign. Lets see an example.

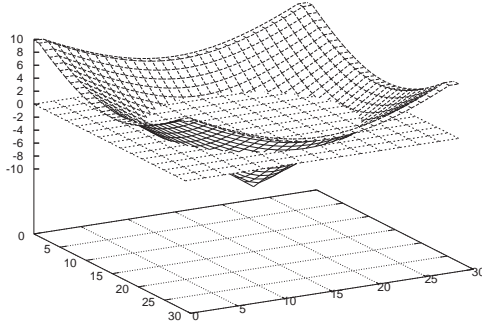


Figure 3.5: Example curve (circle) embedded into a higher dimensional function (signed distance function). Additionally the zero plane is shown.

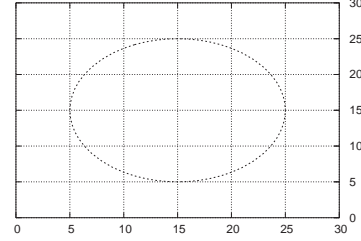


Figure 3.6: The zero isocurve extracted from figure 3.5

Consider the circle curve shown in figure 3.6. The circle curve itself is one dimensional, that is, it can be parametrized using a function taking a one dimensional input value, such as

$$c(t) = (\cos(t), \sin(t)).$$

Where t is bound to be within the interval $[0; 2\pi]$. One problem in handling this function is that it is a vector valued function in the output domain. Lets change that by converting the explicitly defined curve it into an implicit one, called a level set.

In figure 3.5 an implicit representation of the circle is shown. The embedding function ϕ is two dimensional now in its input domain and the one dimensional function value implicitly defines the circle by its zero crossings¹.

We now give reasons why the signed distance function is a good choice as embedding function. Some of the reasons will be discussed in detail in the later sections.

1. It's definition is unique.

Except for the sign, that is if you assume the interior of the interface to be negative or positive, for every given interface, there is exactly one signed distance function.

2. $|\nabla\phi| = 1$ almost everywhere.

Except for points where no gradient is defined the length of the gradient vector is always one. This is an important in that it simplifies definitions of geometric properties such as the curvature.

3. Geometric properties easily derived.

The gradient, normal vector and curvature of the interface is easily derived from the values of the function ϕ .

¹Note that while by convention it is $\phi > 0$ outside the interface, it is perfectly legal to flip the sign of the function.

4. Evolution of the interface can conserve all properties.

Except for small discretization errors the function ϕ can be evolved conserving its signed distance property everywhere.

5. Can be easily (re)constructed.

Given any closed interface the signed distance function can be easily constructed. Particularly well suited to efficiently construct ϕ is the Fast Marching Method. Reconstructing a new signed distance function from a quality deteriorated one is also possible, in case the function values are still of sufficient quality around the interface itself. This is used in the narrowband level set method.

6. Extends naturally to higher dimensions.

The signed distance function exists for any natural dimension and hence interfaces of any dimension can be used with it.

For this reasons, most modern implementations of the level set method use the signed distance function to embed the interface.

3.2.3 Geometric properties

In most applications of the level set method, geometric properties of the embedded interface relate to important properties in the application domain. As such, it is desirable to know the definitions of common geometric properties. While most of them are available rather directly in the level set method, sometimes it is necessary to first convert the implicit representation into an explicit one. For example, to measure the length of a closed curve accurately, one could apply an isocontour plotting algorithm on the embedding function and then easily measure the length in the resulting plotted curve. We now discuss the geometric properties that are easily available in the implicit representation of the interface. In section 3.3.4 we give a isocontour plotting algorithm for closed curves embedded into a 2D signed distance function.

The normal vector

The normal vector \vec{N} is defined as

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|},$$

where

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y} \right)$$

is the gradient of the embedding function. The gradient always points in the direction of increasing values of ϕ perpendicular to the interface contour. In the level set method, the gradient and hence the normal vector are defined for the entire computational domain, even extending away from the interface.

Assume we want to locate the nearest point on the interface from a given point. When using the signed distance function, given any point (i, j) in the domain, we can obtain the nearest point (p, q) on the interface by subtracting the product of the distance $\phi_{i,j}$ to the interface with the normal $N_{i,j}$ at the point from it:

$$(p, q) = (i, j) - \phi_{i,j} \cdot \vec{N}_{i,j}.$$

However, this definition is not unique. There are points for which there are two or more points at the interface having the same shortest distance. In fact, the points which have more than one unique nearest neighbor at the interface make up the skeleton of the interface². In a numerical implementation this can be a problem as a finite difference approximated gradient is inaccurate at such points or – in the extreme case – does not even exist. Therefore, we now examine finite approximations of the gradient and second order derivatives.

Finite approximations

Following Sethian [46], we will now discuss the basic idea of implementing the level set method by means of finite approximations. So far we have assumed a continuous computational domain in both spatial and time dimension, which – while appealing in theory – is not realizable in practice. By discretizing the continuous space we can implement it in a computing system. However, the discretized version is just an approximation, and attention must be paid to what continuous properties are lost and how the numerical quality of the solution is diminished.

Commonly the computational domain is two or three dimensional in space plus one time dimension. For simplicity let's assume a two dimensional space here. The spatial dimensions are discretized by ordered points (x, y) on a grid, where the distance between the points is horizontal Δx and vertical Δy . Most often $\Delta x = \Delta y$. Similarly the continuous time is discretized in time steps Δt .

We now show how the gradient $\phi_x(x, y, t)$ can be approximated by means of finite difference methods. The idea is to use the first terms of a Taylor series to obtain an approximation to the gradient. We first consider the expanded Taylor series to obtain $\phi(x + \Delta x, y, t)$:

$$\phi(x + \Delta x, y, t) = \phi(x, y, t) + \phi_x(x, y, t)\Delta x + O(\Delta x^2)$$

To obtain the gradient we are interested in, we rearrange the equation to get

$$\phi_x(x, y, t) = \frac{\phi(x + \Delta x, y, t) - \phi(x, y, t)}{\Delta x} - O(\Delta x^2)$$

and drop the $O(\Delta x^2)$ term³, which contains all the second order and higher order terms of the Taylor series.

$$\phi_x^{+x}(x, y, t) = \frac{\phi(x + \Delta x, y, t) - \phi(x, y, t)}{\Delta x} \approx \phi_x.$$

This approximation is called *forward difference* approximation, and there also exist a *backward difference* approximation as

$$\phi_x^{-x}(x, y, t) = \frac{\phi(x, y, t) - \phi(x - \Delta x, y, t)}{\Delta x} \approx \phi_x.$$

A more accurate approximation is the *centered difference* scheme which takes the average between the forward and backward difference approximations as

$$\phi_x^{0x}(x, y, t) = \frac{\phi_x^{+x}(x, y, t) + \phi_x^{-x}(x, y, t)}{2} = \frac{\phi(x + \Delta x, y, t) - \phi(x - \Delta x, y, t)}{2\Delta x} \approx \phi_x.$$

²Which is strictly true only for points within the interface, but there is a similar analogy outside of it.

³There are also higher order schemes that use more terms of the Taylor series. One of them is the *Lax-Wendroff scheme*.

Similarly the second derivative can be approximated as centered difference scheme with

$$\phi_{xx}^{+x-x} = \frac{\phi(x + \Delta x, y, t) - 2\phi(x, y, t) + \phi(x - \Delta x, y, t)}{\Delta x^2} \approx \phi_{xx}. \quad (1)$$

The other spatial dimensions are defined in the same way and the derivative over time is approximated using the same first order forward difference expansion, as

$$\phi_t = \frac{\phi(x, y, t + \Delta t) - \phi(x, y, t)}{\Delta t} - O(\Delta t).$$

With so many choices to approximate ϕ_x , which shall we choose? The main concerns are stability and accuracy. Because we use a finite grid, singularities such as sharp corners are not well represented and by using finite difference approximations to obtain the derivative, they affect the derivative of neighbor grid points. Informally speaking, in such situations it is preferable to abandon accuracy in favor of choosing the finite approximation that bases its resulting value on smooth parts of the domain away from singularities that cause problems. We will discuss this concept more formally later, when introducing *upwind schemes* in section 3.3.2, where we will also discuss more advanced approximation methods to achieve higher order accuracy and how the time derivative is used in the evolution equation.

Curvature

A good introduction about the various means to measure curvature and theorems relating to them can be found in [16]. Here we are concerned with the curvature in the level set framework. We first give a short intuitive description of the meaning of curvature and then formally show how it can be derived when using level sets.

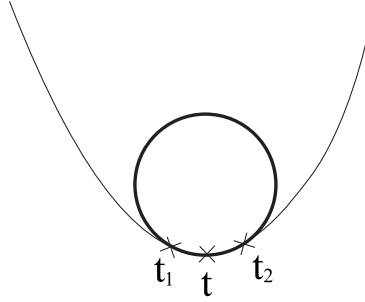


Figure 3.7: Osculating circle defining the curvature on a curve.

Intuitively, the curvature is a local property of a curve or surface which describes how “bend” the curve or surface is around a point. Visually for a curve, this can be described using the concept of the *osculating circle*. Consider the curve shown in figure 3.7. Let $C(t_1, t, t_2)$ be a circle through the three non-identical points t_1 , t and t_2 , where all the points lie on the curve. Then, the osculating circle is the unique circle C , where

$$C = \lim_{t_1, t_2 \rightarrow t} C(t_1, t, t_2).$$

Visually, in highly twisted parts of the curve this circle is small. In near linear parts the circle is becoming large. For the special case of a line, the circle radius goes to infinity. Formally,

	1	
1	-4	1
	1	

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

Figure 3.8: Three commonly used discrete approximations to the Laplacian.

the radius r of the osculating circle relates to the curvature $\kappa(t)$ at the point t by

$$r = \frac{1}{|\kappa(t)|}$$

The sign of the curvature is defined depending on which side of the curve the circle is located. It is chosen so convex parts of the interface have a positive curvature and concave parts have a negative one. For the three dimensional case, there is more than one measure of curvature. We now examine the curvature for the two dimensional case in the level set framework.

Formally, the curvature is defined on the normal $\vec{N} = (n_1, n_2)$ of an interface as

$$\kappa = \nabla \vec{N} = \frac{\partial n_1}{\partial x} + \frac{\partial n_2}{\partial y}.$$

Further, when using the signed distance function as embedding function we know that except for a few boundary cases we have $|\nabla \phi| = 1$. So we have

$$\kappa = \nabla \cdot \vec{N} = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right) = \nabla \cdot (\nabla \phi) = \Delta \phi = \phi_{xx} + \phi_{yy}.$$

So the beautiful result of embedding the interface into the signed distance function is that the curvature at the points in the domain is identical to the Laplacian at that position. From a practical viewpoint this simplifies the implementation of the level set method, as there are well known discrete approximations for the Laplacian which can be implemented using convolution. Three commonly used approximations from [15] are shown in figure 3.8.

To show how these approximations relate to the formulas of the curvature above, we derive the first approximation from finite difference gradients. We first have

$$\kappa = \phi_{xx} + \phi_{yy}.$$

Now, assuming a discretized domain with a spatial resolution of Δx and Δy , we can use the simple finite difference approximations to the first and second order derivatives as in equation 1. Expanding each of the second derivatives for a single point at (i, j) yields

$$\kappa_{i,j} = \frac{\phi(i + \Delta x, j) - 2\phi(i, j) + \phi(i - \Delta x, j)}{\Delta x^2} + \frac{\phi(i, j + \Delta y) - 2\phi(i, j) + \phi(i, j - \Delta y)}{\Delta y^2}.$$

Usually we have $\Delta x = \Delta y$, so we can combine the terms to

$$\kappa_{i,j} = \frac{\phi(i + \Delta x, j) + \phi(i, j + \Delta y) - 4\phi(i, j) + \phi(i - \Delta x, j) + \phi(i, j - \Delta y)}{\Delta x^2},$$

which corresponds to convolution using the first approximation of the Laplacian in figure 3.8 and a division by Δx^2 , the squared spatial step size⁴.

⁴Another way to say this is that the Laplacian convolution masks implicitly assume $\Delta x = \Delta y = 1$.

The curvature representable using a discretized domain of a spatial step size Δx is limited by

$$-\frac{1}{\Delta x} \leq \kappa \leq \frac{1}{\Delta x}.$$

For practical purposes, values exceeding this range are set to the boundary values, as explained in [39]. It makes no sense to define a higher absolute curvature than the boundary values because the implicit representation cannot model structures smaller than Δx .

The Laplacian is sensitive to noise in the embedding function ϕ . If ϕ is noisy, derivatives of ϕ will be even noisier. As the curvature makes use of second derivatives, it is even more noisier [39].

3.3 Ingredients

To work with and implement level set methods, there is a basic “toolbox” of functions required. Here I give descriptions and implementations to the algorithms I use. In detail, the following methods are explained.

- *Initialization* of the level set function from a given parametric contour.
- *Reinitialization* of a partially deteriorated level set function.
- *Evolution* of a level set function under given evolution equations.
- *Hamilton-Jacobi Essentially-Non-Oscillatory schemes (HJ ENO)* for high order spatial approximation to the gradient of the level set function.
- *Isocontour extraction algorithms* to obtain a parametrized contour from a given level set function.

3.3.1 Initialization/Reinitialization methods

The level set function ϕ represents the interface and is the structure the evolution methods modify. To provide the levelset function for an arbitrary interface, we need an *initialization method* which can construct the levelset function from an arbitrary explicit interface representation. Equally important is the *reinitialization function*, which can convert a quality-wise deteriorated levelset function into a “repaired” one. The latter is important for two reasons, (a) numerical approximation errors could degrade the signed distance property locally, and (b) in the narrowband levelset method, we need to advance the narrowband every few time steps, providing a new signed distance function.

Initialization from an explicit interface

For the initial construction of the signed distance function ϕ from an explicit interface, we need to answer only two questions for every point (x, y) in the computational domain:

1. Is (x, y) inside or outside of the interface?

The answer to this question determines the sign of the resulting function value $\phi(x, y)$. By convention $\phi(x, y) < 0$ for points inside the interface, and $\phi(x, y) > 0$ outside of it.

2. What is the distance of (x, y) to the nearest point on the interface?

The distance is just the absolute function value $|\phi(x, y)|$, namely, the normal distance function.

The following pseudo code algorithm works on any consistent explicit interface representation.

Algorithm *InitializeLevelset*

Input: The set of explicit interface elements S . The dimensions of the computational domain.

Output: A new signed distance levelset function ϕ for the interface.

1. **for** (x, y) in the computational domain
2. $minDist \leftarrow +\infty$
3. **for** $e \in S$
4. $minDist \leftarrow \min\{minDist, dist(e, (x, y))\}$
5. **if** $IsInsideInterface(x, y)$
6. $minDist \leftarrow -minDist$
7. $\phi(x, y) \leftarrow minDist$
8. **return** ϕ

The algorithm is straightforward. For every point in the domain, the minimum distance to the interface is determined in lines 2 to 4. The function $dist(e, (x, y))$ depends on how the explicit interface is specified. For example, in polygon based approximation of a curve, this could be the minimum distance from all the line points to (x, y) . To not explicitly specify this function allows the algorithm to work on complex interface representations. In line 5 the sign is made negative in case the point is inside the interface. Again the predicate $IsInsideInterface(x, y)$ is left unspecified so it could work in a general way.

Reinitialization

Reinitialization of the level set function ϕ means restoring the signed distance function without moving the underlying interface, as summarized in [25]:

This is exactly what is desirable for reinitialization: an equation that finds the distance function from the zero level set without moving the zero level set itself and without knowing explicitly the position of the zero level set.

There is a tradeoff between how well the signed distance property is restored and how far the interface is moved. For example, consider the case if ϕ only roughly approximates the signed distance function, even around the contour. Then, only approximately $|\phi(x, y)| = 1$ around the zero crossings. If we choose to perfectly restore the signed distance property, we would have to change the values of $\phi(x, y)$ even on points very close to the interface, which would move the locations of the zero crossings. Instead, if we choose to conserve the values $\phi(x, y)$ nearby the interface, we do not restore $|\phi(x, y)| = 1$ at those places. We choose to implement the later approach, although there are more advanced schemes discussed in detail in [25].

The simple scheme we use does not move the interface at all and approximates the signed distance function in first order accuracy⁵.

⁵However, “not moving the interface” depends on how to locate the interface in the first place. If a high order accurate method is used to extract the interface from ϕ , it may appear to have moved after reinitialization with our scheme, too. If a simple one-grid stencil is used to locate it, as in the scheme given in section 3.3.4, it will not move.

We now give pseudo code to reinitialize a fully defined levelset function $\phi(x, y)$ so the position of the isocontour is conserved while the signed distance function condition is restored everywhere.

Algorithm *ReinitializeLevelset*

Input: The discretized levelset function ϕ to be reinitialized.

Output: A new signed distance levelset function ϕ' having the same contour as ϕ .

```

1.  $KNOWN_1 \leftarrow \{\}$ 
2.  $KNOWN_2 \leftarrow \{\}$ 
3. for  $(x, y)$  in  $\phi$  except the right lower border elements
4.     if  $\text{sign}(\phi(x, y)) \neq \text{sign}(\phi(x + 1, y))$ 
5.          $KNOWN_1 \leftarrow KNOWN_1 \cup \{(x, y, \phi(x, y)), (x + 1, y, \phi(x + 1, y))\}$ 
6.          $KNOWN_2 \leftarrow KNOWN_2 \cup \{(x, y, -\phi(x, y)), (x + 1, y, -\phi(x + 1, y))\}$ 
7.     if  $\text{sign}(\phi(x, y)) \neq \text{sign}(\phi(x, y + 1))$ 
8.          $KNOWN_1 \leftarrow KNOWN_1 \cup \{(x, y, \phi(x, y)), (x, y + 1, \phi(x, y + 1))\}$ 
9.          $KNOWN_2 \leftarrow KNOWN_2 \cup \{(x, y, -\phi(x, y)), (x, y + 1, -\phi(x, y + 1))\}$ 
10.  $KNOWN_1 \leftarrow \text{EvolveFMM}(KNOWN_1, F(x, y) = 1)$ 
11.  $KNOWN_2 \leftarrow \text{EvolveFMM}(KNOWN_2, F(x, y) = 1)$ 
12.  $\phi' \leftarrow \{\}$ 
13. for  $(x, y)$  in  $\phi$ 
14.     hasElement  $\leftarrow \text{false}$ 
15.     for  $(x, y, v) \in KNOWN_1$ 
16.          $\phi'(x, y) \leftarrow v$ 
17.         hasElement  $\leftarrow \text{true}$ 
18.     if hasElement = false
19.         for  $(x, y, v) \in KNOWN_2$ 
20.              $\phi'(x, y) \leftarrow -v$ 
21.             hasElement  $\leftarrow \text{true}$ 
22.     if hasElement = false
23.         abort
24.  $\phi' \leftarrow \text{FixBorder}(\phi')$ 
25. return  $\phi'$ 

```

The *ReinitializeLevelset* procedure works in three steps.

1. Zero crossing element identification (lines 3 to 9).

In both horizontal and vertical direction the zero crossing elements are extracted. A zero crossing is present between $\phi(x, y)$ and $\phi(x + 1, y)$ or $\phi(x, y)$ and $\phi(x, y + 1)$ if the signs are not equal. The function values and their coordinates are stored in two sets, $KNOWN_1$ and $KNOWN_2$, where the sign is flipped for $KNOWN_2$.

The geometric interpretation of these two sets is given in figure 3.9, in which only the x direction is drawn and $\phi(x, y) < 0 < \phi(x + 1, y)$. Then, the outside of the interface is in direction of increasing x-coordinates, and we store the two values in the $KNOWN_1$ set. By reversing the sign of both values and adding them to the $KNOWN_2$ set, the following FMM propagation step will walk inward. The idea is to have two unsigned distance functions, one for the inside and one for the outside of the interface. Note that by reversing the sign, we do not modify the interface position.

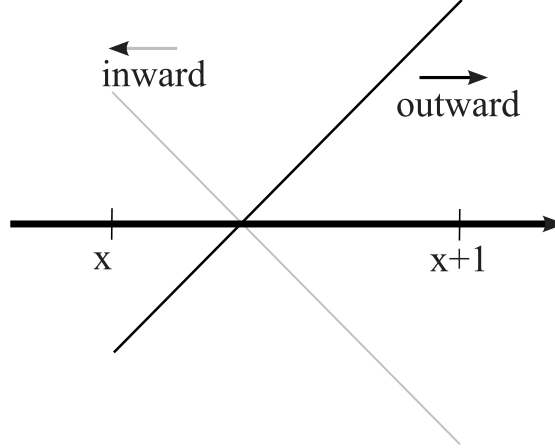


Figure 3.9: Constructing two KNOWN sets from the zero crossing elements.

2. Building two distance functions using FMM (lines 10 and 11).

From the given KNOWN sets, the Fast Marching Method is used to evolve the given known values with a unit normal speed $F(x, y) = 1$ everywhere. This effectively constructs the unsigned distance function for the outside ($KNOWN_1$) and the inside ($KNOWN_2$) of the curve.

3. Merging two unsigned distance functions to one signed distance function (lines 12 to the end).

The resulting $KNOWN_1$ and $KNOWN_2$ sets are merged into one signed distance function ϕ' . Except for the zero crossing elements, $KNOWN_1$ and $KNOWN_2$ share no elements. Together they cover the entire computational space, which is checked in line 22.

3.3.2 Evolution methods

The general *level set equation* capturing the evolution of level sets is

$$\phi_t + \vec{V} \cdot \nabla \phi = 0, \quad (2)$$

where ϕ_t denotes the partial derivative of the embedding function ϕ over time. Equation 2 is a partial differential equation that defines the motion of the interface as a property of ϕ , namely $\phi(x, y) = 0$. \vec{V} denotes a continuous velocity field defined in the computational domain and deserves some attention.

The velocity field \vec{V}

The velocity field determines the motion of the interface in the computational domain. The velocity depends on the specific problem to be solved and can often be easily derived for points on the interface. But for the use of equation 2 we need \vec{V} to be defined in the *entire* computational domain. In fact, not only the interface is embedded into the computational domain, but also the interface velocity needs to be defined on the higher dimensional domain.

In many evolution problems there is a meaningful way to derive the velocity away from the interface. This is the case for image segmentation, as the embedding domain has the same dimensionality as the original image and the velocity is derived from the image. For some cases, such relationship cannot be found easily. Then, the *extension velocity* away from the interface can be used, which states that for every point in the computational domain the velocity at the nearest interface point is used [60]. The method has the advantage that the signed distance property is conserved and the extension velocity can be found efficiently everywhere in the domain. A numerical evaluation of the accuracy is given in [9].

In order to discretize equation 2, we have to examine the dependency of the velocity term and its effect on the interface. We follow the excellent discussion in [39].

Hyperbolic terms

Lets first assume a given velocity field independent of the interface itself, where the velocity \vec{V} is known at every point in the domain. In order to evolve the levelset function ϕ over time, we can discretize equation 2 over time using the first-order accurate *forward Euler* method [61] to obtain

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \vec{V}^n \cdot \nabla \phi^n = 0, \quad (3)$$

where Δt is one time step, ϕ^{n+1} is the levelset function at time step $n + 1$, and \vec{V}^n is the velocity field at time n . Without going into depth about the reasons, the discretization of the hyperbolic equation 3 requires that only information from the direction which determines the outcome of the equation is used⁶. Lets see how to solve for ϕ^{n+1} . If $\vec{V}^n = (u^n, v^n)$ is the vector field, equation 3 can be rewritten as

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n \phi_x^n + v^n \phi_y^n = 0, \quad (4)$$

with ϕ_x^n, ϕ_y^n being the spatial derivatives of ϕ at time step n . For every dimension these derivatives need to be approximated directionally according to the *method of characteristics*. For example, consider a positive u^n , which means the values of ϕ move from left to right. Then, in order to approximate the component ϕ_x^n , the method of characteristics tells us to look into the direction of the origin, that is to the left of the values in ϕ^n , to determine the outcome ϕ^{n+1} .

To implement the evolution equation, the time discretized form in equation 4 also needs to be discretized in the spatial dimensions, which is straightforwardly done by using a Cartesian grid with a step size of Δx and Δy to obtain for every grid point (i, j) :

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^n}{\Delta t} + u_{i,j}^n (\phi_x)_{i,j}^n + v_{i,j}^n (\phi_y)_{i,j}^n = 0. \quad (5)$$

The derivatives $(\phi_x)_{i,j}^n$ and $(\phi_y)_{i,j}^n$ at the grid points have to be approximated using finite difference methods, as shown in section 3.2.3. Back then, we had more than one stencil to choose from to approximate the derivative. Which stencil we choose now depends on the sign of $u_{i,j}^n$ and $v_{i,j}^n$ respectively. The resulting method is called *upwinding*.

⁶In the context of this thesis we will not discuss the numerical discretization of partial differential equations, but only give the schemes necessary to implement the levelset method. For a more detailed discussion see [39, 46, 61]

Now we can straightforwardly solve equation 5 for $\phi_{i,j}^{n+1}$ to obtain

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t [u_{i,j}^n (\phi_x)_{i,j}^n + v_{i,j}^n (\phi_y)_{i,j}^n]. \quad (6)$$

Parabolic terms

We have just examined the evolution for a given velocity field. In most applications the velocity the interface is moved with depends on the interface itself. A common case is moving the interface under the influence of its curvature. We now discuss the discretization of these cases, which are *parabolic equations*.

Lets assume a velocity field \vec{V} depending on the interface, where the velocity at every point can be given as $(V_n \vec{N} + V_t \vec{T})$, the sum of the velocity in normal direction to the front V_n plus the tangential velocity V_t . Clearly, by evaluation of equation 2 we yield

$$\phi_t + (V_n \vec{N} + V_t \vec{T}) \cdot \nabla \phi = 0, \quad (7)$$

with the product $V_t \vec{T} \cdot \nabla \phi = 0$ everywhere, as $\vec{T} \cdot \nabla \phi = 0$. Hence, we only have to consider $V_n \vec{N}$, the velocity normal to the interface. The term $\vec{N} \nabla \phi$ can be reduced:

$$\vec{N} \cdot \nabla \phi = \frac{\nabla \phi}{|\nabla \phi|} \cdot \nabla \phi = |\nabla \phi|. \quad (8)$$

Combining 7 and 8, the simplified evolution equation is

$$\phi_t + V_n |\nabla \phi| = 0, \quad (9)$$

which is also referred to as the levelset equation. In the usual case the *normal velocity* depends on the second derivative of ϕ , as is the case if it contains a curvature dependent term. Then, the term is a *parabolic term* and we cannot use the upwinding scheme of the previous section. Instead, we need a discretization of the second derivative that includes information from all directions. We have given such a finite difference stencil in section 3.2.3. Because we use a signed distance function, $|\nabla \phi| = 1$ and so the term can be dropped. Otherwise the discretization is the same as for hyperbolic terms and the final evolution equation is

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t [(V_{i,j})_n]. \quad (10)$$

One has to be aware though, that when using this Euler time discretization, the resulting function may not be a signed distance function anymore. To further evolve it using equation 11, the function has to be reinitialized again. For the common case of $(V_{i,j})_n = \alpha \kappa_{i,j}$, with $\kappa_{i,j}$ being the curvature as defined in section 3.2.3, where $\kappa_{i,j} = \Delta \phi_{i,j}$, equation 11 becomes

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t (\alpha \kappa_{i,j}) = \phi_{i,j}^n - \Delta t (\alpha \Delta \phi_{i,j}). \quad (11)$$

Convergence

For the finite difference approximations used above to be a usable approximation, they have to converge to the correct solution of the partial differential equation, namely the solution to the levelset equation. The given schemes above are convergent if and only if it is *consistent*

and *stable*⁷. We have to examine both the upwind scheme and the given approximation to the parabolic term for convergence.

1. Upwind scheme.

The given upwinding scheme uses Euler time discretization. This makes it consistent, as for $\Delta t \rightarrow 0$ and $\Delta x, \Delta y \rightarrow 0$ the approximation error converges to zero. What is left to examine is stability, the guarantee that small approximation errors are not amplified over time. The *Courant-Friedrichs-Lewy* (CFL) condition provides a time step restriction of

$$\Delta t < \frac{\min \{\Delta x, \Delta y\}}{\max \{|u|\}},$$

where $\max \{|u|\}$ is the maximum velocity in the computational domain. Using appropriate values of Δt , Δx and Δy , this makes the upwinding scheme stable, hence convergent.

2. Parabolic central differencing scheme.

Using the central differencing scheme, the above scheme to solve parabolic terms is still consistent. Stability however cannot be achieved anymore using the above CFL condition. For $V_n = \alpha\kappa$ it can be achieved using the stricter CFL condition of

$$\Delta t \left(\frac{2\alpha}{(\Delta x)^2} + \frac{2\alpha}{(\Delta y)^2} \right) < 1$$

Unfortunately, because of the much smaller time step the additional computational effort is high. There are two alternative methods to make the scheme stable without using this very strict CFL condition. (a) Using higher order temporal discretization schemes⁸, or (b) introducing an artificial dissipation term into the levelset equation. The discussion of these two techniques are out of the scope of this thesis.

Algorithmic Implementation

We now give a pseudo code implementation of the evolution algorithm that solves one time step of the evolution equation. It uses Euler time discretization, upwinding for hyperbolic terms and also allows parabolic terms in the speed function. The speed function $F(\phi, x, y)$ is represented as the sum of parabolic and hyperbolic terms:

$$F(\phi, x, y) = F_{parabolic}(\phi, x, y) + F_{hyperbolic}(\phi, x, y).$$

The algorithm only deals with a two dimensional ϕ , but can be easily extended to handle an arbitrary number of dimensions.

Algorithm *EvolveLevelset*

Input: The levelset function ϕ , the time step Δt . The speed function terms $F_{hyperbolic}$ and $F_{parabolic}$, where one of them could be undefined. $F_{hyperbolic}$ returns a directional vector (s_x, s_y) the interface is moved with, whereas $F_{parabolic}$ is expected to return the speed in normal direction.

⁷As explained in [39] with reference to the Lax-Richtmyer equivalence theorem.

⁸Such as Runge-Kutta temporal discretization.

Output: A new levelset function ϕ' . In case $F_{parabolic}$ is used, its not guaranteed to be the signed distance function.

```

1. for  $(x, y)$  in  $\phi$  except the border elements
2.      $c_{hyper} \leftarrow 0$ 
3.      $c_{para} \leftarrow 0$ 
4.     if  $F_{hyperbolic}$  is defined
5.          $(s_x, s_y) \leftarrow F_{hyperbolic}(\phi, x, y)$ 
6.         if  $s_x \geq 0$ 
7.              $g_x \leftarrow calculateDerivativeBackwardX(\phi, x, y)$ 
8.         else
9.              $g_x \leftarrow calculateDerivativeForwardX(\phi, x, y)$ 
10.        if  $s_y \geq 0$ 
11.             $g_y \leftarrow calculateDerivativeBackwardY(\phi, x, y)$ 
12.        else
13.             $g_y \leftarrow calculateDerivativeForwardY(\phi, x, y)$ 
14.         $c_{hyper} \leftarrow s_x \cdot g_x + s_y \cdot g_y$ 
15.        if  $F_{parabolic}$  is defined
16.             $c_{para} \leftarrow F_{parabolic}(\phi, x, y)$ 
17.         $\phi'(x, y) \leftarrow \phi(x, y) - \Delta t \cdot (c_{para} + c_{hyper})$ 
18.  $\phi' \leftarrow FixBorder(\phi')$ 
19. return  $\phi'$ 

```

The algorithm *EvolveLevelset* is straight forward by handling the hyperbolic and parabolic terms separately. Starting at line 4 the hyperbolic speed function $F_{hyperbolic}$ is evaluated, which returns a velocity vector (s_x, s_y) for the position. The components of this vector determines the movement direction for each dimension, which in turn defines the upwinding direction. Up to line 14 the gradient is approximated using the appropriate upwinding direction, and the product $\vec{V} \nabla \phi$ is calculated, which determines the movement in normal direction c_{hyper} stemming from the hyperbolic term. Note that we do not explicitly specify how the gradient is approximated, and any upwinding scheme can be used here⁹.

The calculation of the parabolic component c_{para} starting from line 15 using the speed term $F_{parabolic}$ is more straightforward and can be evaluated directly. In line 17 the two terms are combined and the Euler time discretization is used to obtain the next value in ϕ' .

3.3.3 HJ ENO scheme

In this section we explain a high order scheme to approximate the gradient ϕ_x^+ and ϕ_x^- of the levelset function ϕ . Previously we have discussed first and second order finite difference schemes for ϕ_x for use with upwinding schemes and parabolic terms. For upwinding, we can improve upon the accuracy by using a higher order scheme which uses more information around the point to approximate ϕ_x . One such scheme is the Essentially Non-Oscillatory scheme (ENO), invented by Harten and Osher in [20] for conservation laws. Osher and Sethian later realized in [40] that the ENO scheme could be adapted to approximate the gradient in Hamilton-Jacobi (HJ) equations such as equation 2. The simplified scheme here is taken from [39] which points to [48] and [49].

⁹In fact, in our implementation we use a high order ENO scheme, but the given algorithm works as well with the simple finite difference schemes given in section 3.2.3.

In the ENO scheme every partial derivative is approximated by using only values from its respective dimension. Therefore we only consider the one dimensional case $\phi(x)$ and higher dimensional functions ϕ are treated dimension-by-dimension. The idea of the ENO scheme is to approximate $\phi(x)$ as a polynomial

$$\phi(x) = Q_0(x) + Q_1(x) + Q_2(x) + Q_3(x).$$

Where Q_n is a n 'th order polynomial term. When differentiating over x , the approximation becomes

$$\phi_x(x) = Q'_1(x) + Q'_2(x) + Q'_3(x).$$

The polynomial terms are selected from a subset of the function values around the grid point i in the respective dimension d , namely $\{\phi^d(i-3), \phi^d(i-2), \phi^d(i-1), \phi^d(i), \phi^d(i+1), \phi^d(i+2)\}$. The terms Q_1 , Q_2 and Q_3 are build from subsets of these values by choosing the direction which minimizes the absolute gradient. This construction is based on the observation that a large gradient is present in case the function oscillates around the given index i . By choosing the grid points that lead to small absolute gradients, the smoothness of the result polynomial is maximized. We now give a pseudo code algorithm to evaluate $\phi_x^d(i)$, the derivative at grid point i in dimension d , where $d \in \{x, y\}$. $\phi^x(x)$ is a cut of the original function ϕ at a given value of y , so $\phi^x(x) = \phi(x, y)$ for a known y . This allows us to write the algorithm below in a dimension-independent way.

Algorithm *DerivativeENO*

Input: The one dimensional levelset function ϕ^d in dimension d of ϕ , the index i at which the derivative is taken, i_s the direction bias (0 for forward derivative, 1 for backward derivative) and the spatial step size Δx .

Output: The numerical approximation to the derivative $\phi_x^d(i)$.

1. $k \leftarrow i - i_s$
2. $d \leftarrow D(\phi^d, k, 1, \Delta x)$
3. **if** $|D(\phi^d, k, 2, \Delta x)| \leq |D(\phi^d, k+1, 2, \Delta x)|$
4. $c_1 \leftarrow D(\phi^d, k, 2, \Delta x)$
5. $k_1 \leftarrow k - 1$
6. **else**
7. $c_1 \leftarrow D(\phi^d, k+1, 2, \Delta x)$
8. $k_1 \leftarrow k$
9. $d \leftarrow d + c_1(2(i - k) - 1)\Delta x$
10. **if** $|D(\phi^d, k_1, 3, \Delta x)| \leq |D(\phi^d, k_1+1, 3, \Delta x)|$
11. $c_2 \leftarrow D(\phi^d, k_1, 3, \Delta x)$
12. **else**
13. $c_2 \leftarrow D(\phi^d, k_1+1, 3, \Delta x)$
14. $d \leftarrow d + c_2(3(i - k_1)^2 - 6(i - k_1) + 2)(\Delta x)^2$
15. **return** d

Algorithm *D*

Input: The one dimensional levelset function ϕ^d in dimension d of ϕ , the index j at which the derivative is taken, the finite difference recursion depth k and the spatial step size Δx .

Output: The k 'th divided difference of ϕ^d at index i .

1. **if** $k = 0$ **return** $\phi^d(j)$
2. **if** k is odd
3. **return** $\frac{D(\phi^d, j+1, k-1, \Delta x) - D(\phi^d, j, k-1, \Delta x)}{k\Delta x}$
4. **else**
5. **return** $\frac{D(\phi^d, j, k-1, \Delta x) - D(\phi^d, j-1, k-1, \Delta x)}{k\Delta x}$

In the *DerivativeENO* procedure, the values of Q'_1 (line 2), Q'_2 (lines 3 to 9) and Q'_3 (lines 10 to 14) are approximated and $\phi_x^d(i)$ is approximated in d .

Commonly, divided difference polynomial approximation is used to build the polynomial terms. The divided difference computation is realized in algorithm D . Heath explains this and other polynomial approximations in his introduction to scientific computing [21].

3.3.4 Isocontour extraction

We give a simple isocontour extraction method based on bilinear interpolation. It can be used to extract an explicit line segment representation of a closed curve embedded into a two dimensional signed distance function ϕ . Without loss of generality, the line segments are extracted only for one spatial cell between the coordinates $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. We define

$$a := \phi_{0,0}, \quad b := \phi_{0,1}, \quad c := \phi_{1,0}, \quad d := \phi_{1,1}$$

and the bilinear interpolant inside the cell as

$$\phi'_{x,y} := (c - a)x + (b - a)y + (d + a - b - c)xy + a$$

By solving $\phi'_{x,y} = 0$, we obtain two functions $g(x)$ and $h(y)$ as

$$g(x) = y = -\frac{x(c - a) + a}{(b - a) + x(d + a - b - c)} \quad \text{if } (b - a) + x(d + a - b - c) \neq 0$$

$$h(y) = x = -\frac{y(b - a) + a}{(c - a) + y(d + a - b - c)} \quad \text{if } (c - a) + y(d + a - b - c) \neq 0$$

A simple scheme is now used to extract line segments. Each cell coordinate axis is discretized into a fixed number of equal sized segments. For each axis, we test the function values of the function taking as input the axis coordinate. For the x-axis, we obtain the point $(x, g(x))$, for the y-axis the point $(h(y), y)$. If the resulting point is within the domain, that is $0 \leq x \leq 1$ and $0 \leq y \leq 1$, the point is kept for that axis. The resulting set of points is connected by line segments. One special case to be considered remains if

$$\text{sign}(\phi_{0,0}) = \text{sign}(\phi_{1,1}) \neq \text{sign}(\phi_{0,1}) = \text{sign}(\phi_{1,0}).$$

This corresponds to two separate parts of the contour within the cell. In this case, we do not generate a line between the two separate parts.

For three dimensional isosurface extraction, the Marching Cubes algorithm [5, 37] is often used.

Although these schemes are sufficient when the output is presented to humans for evaluation, in some cases higher precision algorithms are needed to correctly reflect underlying singularities of the curve or surface. One such scheme based on the ENO scheme discussed in section 3.3.3 is given by Siddiqi et al. in [50].

3.4 The narrow-band level set method

So far we have discussed the discretization of the level set function on the whole computational domain. This is called the *full level set method*, as the whole embedding function ϕ is represented and evolved. In this section we discuss a small change to the above method so that only a part of the embedding function is discretized and evolved. The motivation behind this change is a large complexity reduction, as only the elements in the discretized part have to be evolved.

For interface evolution problems we are interested only in the interface itself. The extension of the interface dimensionality when embedding it into ϕ is done in order to obtain a natural implicit description of the interface. However, by extending the interface the resulting evolution equation also has to be applied to the entire embedding function, not just the part covering the zero level contour. Moreover, the speed function also has to be embedded and calculated for all discrete points. Additional to the problem of the computational complexity, the added resources needed to just store the level set function in the whole domain can be large.

In the *narrowband level set method* the computational domain is split into three disjoint sets at any timestep t :

- The *narrowband* around the zero level contour.

Every discrete point whose distance to the zero level contour is below some given distance d_n is in this set. Intuitively this is a band around the interface of width $2d_n$. Formally, any discrete point p for which $|\phi(p)| < d_n$ is in this set.

- The part outside of the narrowband and outside the interface.

Formally, any point p for which $\phi(p) \geq d_n$ is in this set.

- The part outside of the narrowband and inside the interface.

Formally, any point p for which $\phi(p) \leq -d_n$ is in this set.

The idea of the narrowband method is simply to only consider the first set of points, which lie within the narrowband. As the interface is completely contained within this narrowband, no information about the position of the interface is changed or lost by doing so. The second two sets of points are ignored.

In figure 3.10 the idea is illustrated by showing the three domains around an interface curve. The narrowband is the shaded area around the interface.

It is easy to see that for a large d_n value the narrowband becomes so thick that it covers the entire computational domain. Then, the method is no different from the full levelset method. However, commonly small values are selected, often $2 \leq d_n \leq 10$. Then the narrowband method differs in some ways from the full level set method¹⁰. We will now examine problems that can arise from using the narrowband method.

Problems in the Narrowband Level Set Method

- The interface leaves the narrowband.

¹⁰An analysis of typical narrowband sizes is given by Adalsteinsson in [1]

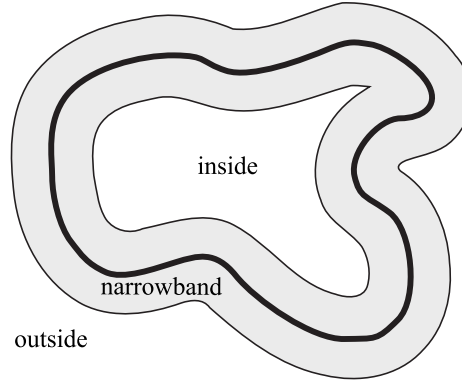


Figure 3.10: Interface with the narrowband neighborhood, inside and outside domains.

If we keep a static narrowband around the interface at one timestep t_0 and evolve the interface over time using the evolution equation, it might leave the narrowband domain.

There are two possible solutions to this problem. The first is to identify the narrowband at every few timesteps k and assign new levelset function values to the narrowband points. This is called reinitialization, and an algorithm similar to the one given in section 3.3.1 is used. The second possible solution is to continuously identify points that join or leave the narrowband and to move the narrowband at every timestep. Because the first solution is well understood and easy to implement it is often preferred.

- The narrowband boundary has no completely defined neighborhood.

For points in the narrowband that lie at the boundary to the inside or outside domain, the function ϕ is not defined for the complete neighborhood. Finite differencing approximations of the gradients run into problems there, as they require at least the direct neighbors in both directions for each dimension to be defined.

A simple solution is to define $\phi(p_o) := d_n$, $\phi(p_i) := -d_n$ for points inside (p_i) and outside (p_o) the narrowband domain. This is similar to the approach taken in defining the boundary of the computational domain in the *FixBorder* algorithm in section 3.5.2. Using this approach, the gradient is always underestimated at the boundary points which smoothes the levelset function during evolution, deteriorating its $|\nabla\phi| = 1$ property over time from the outer points of the narrowband to the middle of the narrowband, where the interface is located. The wrongly estimated gradient information propagates only slowly towards the interface, and as long as we reinitialize the levelset function ϕ every few timesteps the numerical error is minimal.

Implementing the narrowband levelset method is straightforward. From the full levelset method, only the reinitialization algorithm requires changes. Additionally, we have to find a way to efficiently manage the three sets of points.

Reinitialization in the Narrowband Level Set Method. The reinitialization method is nearly identical to the *ReinitializeLevelset* algorithm, with the minor change that the evolution in lines 10 and 11 only proceeds until the distance d_n – half of the narrowband width – is reached in each direction. As the Fast Marching Method iteratively processes one point at a time it is easy to check when this width is reached. All points in the resulting KNOWN set

are marked as being in the narrowband, whereas all other points are in the outside or inside domain.

Handling the sets. For a two-dimensional levelset function ϕ it is sufficient to use a simple labeling scheme, similar to the scheme used in the Fast Marching Method. For three-dimensional embedding functions or for large computational domains it is more efficient to organize the narrowband points in quad-tree and octree datastructures, which are introduced in any algorithm textbook, such as [3, 10].

3.5 Implementation concerns

The algorithms described above provide almost everything that is needed to reimplement a level set framework. For clarity some important details have been omitted so far, and we now take the time to consider them.

1. Discretization.

The discretization of the computational domain has some parameters we can choose, such as the spatial step width Δx and Δy , and the narrow band width in case we use the narrow band method.

2. Handling the boundary.

The discretized domain has a boundary. Some operations, such as approximating the gradient may not be possible on or nearby this boundary, and care is required to not let this limitation diminish the numerical results throughout the evolution of the level set function.

3. Reinitialization frequency.

Reinitialization is computationally costly and should be done as few times as possible. But then, there is a clear lower limit of the reinitialization frequency, below which the evolution method will produce errors.

3.5.1 Discretization

For our level set implementation we assume a Cartesian grid with equal spaced grid points at a step size of $\Delta x = \Delta y$. If the contour in the domain relates to some actual real world contour and its evolution is guided by physical properties, Δx has to be chosen small enough to account appropriately to the underlying physical evolution forces. If chosen too large, places of high curvature will not be represented correctly. However, if the forces guiding the contour evolution are independent of scale, a convenient choice is $\Delta x = \Delta y = 1.0$, as it simplifies most computations¹¹.

The choice of a good narrow band width largely depends on the expected evolution behavior and the resulting need for reinitialization of the narrow band. Equally advancing fronts benefit from a relatively small narrow band width, while fronts that differ a lot in their evolution locally – for example expanding at one place while shrinking at another – benefit from a larger narrow band and less frequent reinitialization. A good generic choice is a width of six elements in each direction.

¹¹Because Δx and Δy commonly appear as denominators.

3.5.2 Handling the boundary

So far, albeit we already discussed the spatial discretization, we practically have assumed a limitless computational domain. Of course, in an actual implementation this is not possible and there has to be a boundary. This introduces a problem when the local neighborhood of a grid point is used to evaluate the gradient or curvature at that point. After all, what is the neighborhood of a grid point on the boundary?

A simple yet in most cases sufficient solution to this problem lies in separating a small read-only boundary around the real computational domain. This boundary is only kept in order to have a neighborhood available for all grid points in the real domain. For all actions on ϕ , only the inner grid points are processed and afterwards the boundary grid points are reinitialized to be copies of their closest inner grid points. Consider the evolution algorithm *EvolveLevelset*, which processes all grid points except the ones lying at the boundary. After processing, it calls the *FixBorder* procedure, which is given below. It restores a one pixel border around the inner computational domain.

Algorithm *FixBorder*

Input: The levelset function ϕ , with a discretized domain $x \in \{0, \dots, x_n\}$, $y \in \{0, \dots, y_n\}$.

Output: A new levelset function ϕ' with the border grid points restored.

1. **for** (x, y) in ϕ except the border elements
2. $\phi'(x, y) \leftarrow \phi(x, y)$
3. **for** x in $\{1, x_n - 1\}$
4. $\phi'(x, 0) \leftarrow \phi(x, 1)$
5. $\phi'(x, y_n) \leftarrow \phi(x, y_n - 1)$
6. **for** y in $\{1, y_n - 1\}$
7. $\phi'(0, y) \leftarrow \phi(1, y)$
8. $\phi'(x_n, y) \leftarrow \phi(x_n - 1, y)$
9. $\phi'(0, 0) \leftarrow \phi(1, 1)$
10. $\phi'(x_n, 0) \leftarrow \phi(x_n - 1, 1)$
11. $\phi'(0, y_n) \leftarrow \phi(1, y_n - 1)$
12. $\phi'(x_n, y_n) \leftarrow \phi(x_n - 1, y_n - 1)$
13. **return** ϕ'

3.5.3 Reinitialization frequency

We have already seen that in most cases reinitialization of the level set function is unavoidable. But how to determine when it is necessary to reinitialize? There are two common methods.

1. Every n steps.

The easiest and most often used method is to reinitialize the level set function every fixed n evolution timesteps. This is adequate for equally advancing contours. We employ this method and reinitialize every 12 or 20 evolution steps.

2. Landmines.

In cases the contour almost stands still or is moving irregularly, a faster way than to reinitialize every few steps is to detect the necessity for reinitialization. To do this, some grid points away from the front by a chosen distance d and $-d$ are marked as *landmine* points.

For every evolution step, the sign of the level set function ϕ at the landmine points is compared before/after the evolution. If ϕ changes its sign at any landmine grid point, the contour has crossed the grid point and the entire level set function ϕ is reinitialized.

The advantage of this method stems from its adaptive detection of change in the contour. To check the landmines is far less computationally expensive than to reinitialize without the need. However, a disadvantage of the landmine method is the additional complexity in setting up and observing the landmines.

Chapter 4

The Fast Marching Method

In this chapter we provide an introduction to the “cousin” of the Level Set Method, namely the Fast Marching Method (FMM). We first contrast the FMM with the Level Set Method and follow up with an in depth algorithmic explanation of the method. We provide pseudo code allowing reimplementing of the method, unlike common literature covering the FMM and give implementation advice.

4.1 Comparison to the Level Set Method

While the Fast Marching Method and the family of Level Set methods share the same origin and overlap in their applications, there are practical and theoretical differences, such as the following.

- Initial Value Problem versus Boundary Value Problem.

In the Fast Marching method, the interface evolution problem is stated so the solution is constructed to be the arrival time at every grid point. In the Level Set perspective, the solution is iteratively evolved in the whole domain, so the solution at every time step are the new values of the embedding function. As such, a Boundary Value Problem is solved during the Level Set evolution, while the FMM solves an Initial Value Problem.

- The speed function F .

In the Fast Marching Method, the speed function has to remain static. One constructs the solution instead of evolving it, hence the notion of time during the progress of the algorithm is a different one. The result is that the speed function $F(x, y)$ can only depend on the grid position, not on time dependent properties or local properties of the interface. This rules out a large class of important speed terms used with Level Sets, such as curvature and other regularization terms.

- The runtime performance.

If a min-heap is used for the Fast Marching Method, the resulting complexity is $O(N \log N)$, compared to $O(N^2)$ for the Narrowband Level Set Method and $O(N^3)$ for the full domain Level Set Method.

To summarize, any problem you can solve with the Fast Marching Method, you can solve with the Level Set method. However, the inverse is not true. The solution process in the

FMM cannot incorporate knowledge of local interface properties, which rules FMM out as a solution to a large class of important problems where the solution has to obey some boundary conditions, most importantly smoothness.

4.2 The method

The Fast Marching method is an algorithm to efficiently solve curve and surface evolution problems. Consider a closed curve that evolves under a fixed-sign speed $F(x, y)$ in normal direction dependent only on the position (x, y) in the computational domain. The curve either expands outward all the time or moves inward all the time and once a point has been crossed by the curve it will never be crossed again. Then, the *Eikonal equation* can be given as

$$|\nabla T|F = 1$$

where $T(x, y)$ is the arrival time at which the surface crosses the given point (x, y) . The Eikonal equation states the relationship that the gradient of arrival time is inversely proportional to the speed of the propagating curve [32]. The Fast Marching Method explicitly constructs the solution to $T(x, y)$ for all points (x, y) in the domain by marching away from known solution values, constructing new solutions on the way. The complexity for N points is $O(N \log N)$ ¹.

Recently, a variation of the Fast Marching Method has been proposed in [59] by Yatziv et. al, that makes use of an *untidy priority queue* that only approximates the monotonicity of the min-heap. This allows for a fast implementation in $O(N)$ running time, while virtually has no effect on the numerical accuracy². Here we only consider the classic Fast Marching Method with a tidy heap.

There is a close relationship between the Fast Marching Method and the Dijkstra shortest path algorithm. In fact, the Fast Marching Method is an adaption of the Dijkstra algorithm, on the one hand (a) limiting it by explicitly specifying the structure of the graph to be searched in, and on the other hand (b) extending it by giving the search process a geometric interpretation. The graph to search in is a regular Cartesian grid of points (x, y) , with edges connecting adjacent grid points³. Every point (x, y) has a propagation speed value $F(x, y)$ associated with it. One could interpret $F^{-1}(x, y)$ as a cost function for traveling through the point. Given initial front arrival times $T(x, y)$ for one or more of the grid points, the FMM iteratively constructs $T(x, y)$ for all remaining points in the domain. We now describe this iteration procedure in detail.

4.2.1 Marching step

The Fast Marching Method's name is based on its iterated processing, which is called "down-wind marching". In this section we explain the marching algorithm in detail, using the Fast Marching update rule and the min-heap data structure.

¹And besides the low complexity in Big-O notation, it can be implemented efficiently on today's computing systems.

²The reason they give why this does not diminish the accuracy as much as one would expect is that the order of accuracy loss in the time dimension is less or in the same order of magnitude compared to the spatial discretization.

³We describe the original Fast Marching Method, in [46] the method is extended to allow triangulated domains.

Downwind marching is the process of constructing the solution to the $|\nabla T|F = 1$ equation iteratively from small to large arrival times. Consider the situation shown in figure 4.1. The single dot in the center represents the front in the grid. For this point P_0 we know the arrival time of the front, say $T_0 = 0$. To advance the front from its positions in the grid, we have to consider the neighborhood around all the points belonging to the front and compute their arrival times. In figure 4.2 we show the neighbors of the front.

The crux of the FMM lies in the order the neighbors are considered. In the Fast Marching Method they are processed in the order the front arrives at them. To achieve this, the points are divided among three disjoint sets, *KNOWN*, *TRIAL* and *UNKNOWN* points. All the points with known arrival times, drawn black in the figure, are in the *KNOWN* set. All the points that are in the four neighborhood of *KNOWN* points are in the *TRIAL* set, drawn in shaded gray. Finally, all other points, which are far away from the front are in the *UNKNOWN* set, drawn in white. The method works by assigning each point in the *KNOWN* and *TRIAL* set an arrival time T . The arrival times associated with the *KNOWN* set are fixed and cannot change. However, the arrival times for the points in the *TRIAL* set are temporary arrival times and can change.

If we consider all arrival times of the points in the *TRIAL* set, the point with the minimal time is the point that will be crossed next by the front. As the front propagates only outwards, the other points in the *TRIAL* set with higher arrival time values cannot anymore influence the point with the lowest arrival time. Hence we can be sure that, for this single point, the interim arrival time is its real arrival time. Thus the point is moved from the *TRIAL* set to the *KNOWN* set. This is one downwind marching iteration in the Fast Marching Method.

However the front around this point just changed and the neighbors are closer to the advanced front now. Some neighbor points might not be in the *TRIAL* set. To make the *TRIAL* set the set of neighbor points to the front again, all four neighbors of the point that are not in the *KNOWN* set are moved into the *TRIAL* set. Additionally for all 4-neighbors that are in the *TRIAL* set, new temporary arrival times are calculated.

With this basic description, there are three areas left which have to be clarified, (a) how we calculate the temporary arrival times, (b) how we manage the *TRIAL* points efficiently and (c) a formal algorithm of the above description. The first concern is described below as the Fast Marching update rule. The second concern deals with a specific type of heap data structure and is explained in detail below. Now we describe the third area, a formal algorithm for the above description.

Algorithm *Fast Marching Method*

Input: The initialized *KNOWN* and *TRIAL* sets, $T(p)$ for all $p \in \text{KNOWN}$.

Output: The arrival time field T for all points in *KNOWN*.

(* One iteration in the Fast Marching Method *)

1. **while** *TRIAL* $\neq \emptyset$
2. let P_{min} the point with the minimum arrival time T_{min} in *TRIAL*
3. remove P_{min} from *TRIAL* and add it to *KNOWN*
4. $T(P_{min}) \leftarrow T_{min}$
5. **for** n is a 4-neighbor of P_{min} and $n \notin \text{KNOWN}$
6. $T(n) = \text{Update}T(n)$
7. remove n from *UNKNOWN*
8. add n to *TRIAL*
9. **return** T

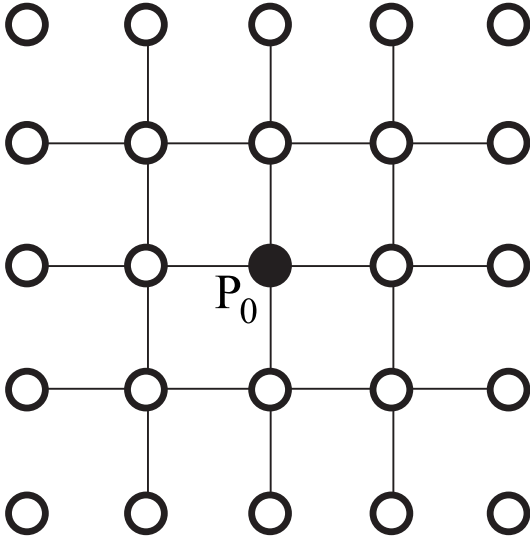


Figure 4.1: FMM Step 1

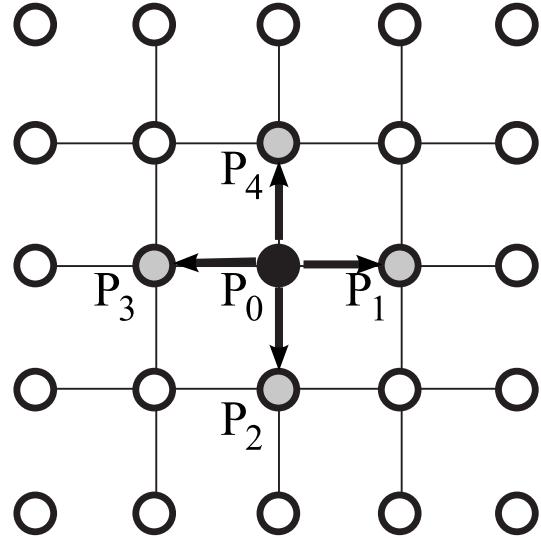


Figure 4.2: FMM Step 2

Lets see how the pseudo code for the *Fast Marching Method* works on the example figure. In figure 4.1 only one point is in the *KNOWN* set. For consistency in the algorithm, we assume the state of 4.1 as input, except that the single point is not in the *KNOWN* set but instead belongs to the *TRIAL* set. The algorithm is started with $KNOWN = \emptyset$, $TRIAL = \{P_0\}$, $T(P_0) = 0$.

As first step P_{min} is identified to be P_0 and $T_{min} = T(P_0) = 0$. P_0 is converted to a *KNOWN* point and removed from the trial set, which reflects the state shown in figure 4.1. Now all the neighbors of P_0 are considered which are not in *KNOWN*, which happen to be all neighbor points, P_1 , P_2 , P_3 and P_4 . They are all added to the *TRIAL* set and a interim arrival time is computed for them. The state is reflected in figure 4.2.

In the second iteration of the main loop, P_2 is identified to be the point with the minimum interim arrival time over the other points P_1 , P_3 and P_4 . P_2 is converted to a *KNOWN* point and removed from the *TRIAL* set. Its interim arrival time is taken as the final arrival time by $T(P_2) = T_{min}$. The neighbors which are not in *KNOWN*, that is P_5 , P_6 and P_7 are converted into *TRIAL* points and an interim arrival time is computed for them. The final step after the second iteration is shown in figure 4.3.

In the third iteration, P_7 happens to be the point with the smallest arrival time $T_{min} = T_7$. It is again converted to a *KNOWN* point and removed from *TRIAL*. When considering the neighbors we have three distinct cases to consider, (a) point P_9 and P_8 are converted from *UNKNOWN* to *TRIAL*, as in the previous two iterations, (b) P_0 is ignored because it is in the *KNOWN* set, a case we already seen so far. But the new case is (c) for P_7 , which is already in the *TRIAL* set. Its interim arrival time value is also updated – decreased in fact – as the front “surrounds” it. The final step after the third iteration is shown in figure 4.4.

While we have a good understanding of the general working of the FMM now, we have not touched upon two critical parts. The first is the construction of the interim arrival time values, which is known as *Update Rule*. The second is how do we efficiently manage the grid points in the *TRIAL* set as to optimize the FMM’s performance, which will lead us to min-heaps. By answering these two questions, we provide the complete recipe for implementing

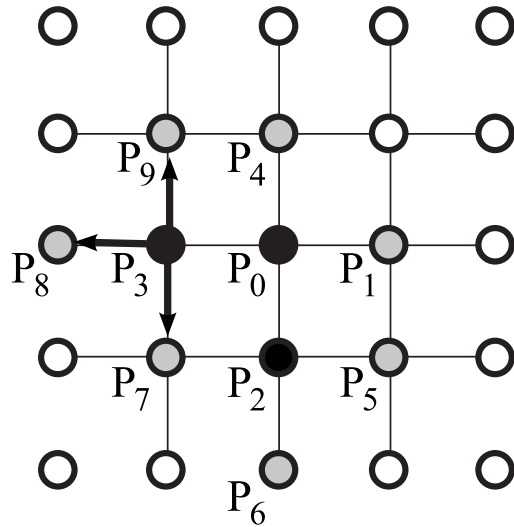


Figure 4.4: FMM Step 4

4.2.2 Update rule

Consider the situation depicted in figure 4.5. The center point P_c is in the TRIAL set and the front has not crossed it so far. It has crossed three neighbors P_0 , P_1 and P_2 , yielding

arrival time values T_0 , T_1 and T_2 , where the current time $T = T_2$, which means the front has just arrived at P_2 , the western neighbor of the considered center point. Given this situation, the problem is to calculate a new interim arrival time T_c for the center point. For this example we assume a speed of $F = 1$ everywhere.

Intuitively, as the front moves in normal direction with equal speed everywhere, we only have to consider the points on the front whose normal vectors extend from the front to the considered point. For the given situation, by only considering the arrival times, at least two such points have to exist. In figure 4.5 they are marked C_0 and C_1 . If we can calculate the distance from these points to the center point, we can calculate a good approximation to a new arrival time.

If the center point has the correct new arrival time, the gradient in the quadrant will have a magnitude of one, as we solve for $|\nabla T|F = 1$, with $F = 1$ to $|\nabla T| = 1$. By knowing which quadrant has the smallest distance from the front to the center point, we only have to consider this quadrant and can ignore all the others. To do this, we first define four quadrants $Q = \{Q_0, Q_1, Q_2, Q_3\}^4$, so Q_0 is marked by the corners P_c , P_0 , and P_1 , the quadrant shaded gray in the figure.

With this definition of quadrants we can select the quadrant containing the point on the front nearest to the center point. For this, in each dimension d we select the neighboring point P_d in the KNOWN set with the smallest arrival time. For each dimension the following three cases can occur:

1. There are no neighbors in the KNOWN set in the given dimension.

If a dimension has no neighboring point in the KNOWN set we simply ignore that dimension. This case does not appear in figure 4.5.

2. There is one neighboring point in the KNOWN set.

Then we always use the point in the KNOWN set. In figure 4.5 this case appears for the y-direction, where P_1 and P_3 are considered. As P_3 is not in the KNOWN set, it is ignored and P_1 is selected.

3. There are two neighboring points in the KNOWN set.

In figure 4.5 this case appears when the x-direction is considered. The points P_0 and P_2 are checked, both of which are in the KNOWN set. Their associated arrival times T_0 and T_2 are compared and P_0 is selected as $T_0 < T_2$, so T_0 is the minimal arrival time.

In general, let P be the set of neighboring points that have been collected. For n dimensions $1 \leq |P| \leq n$. In case $|P| = n$, the points in P mark the quadrant with the closest front. For two dimensions in case $|P| = 1$, the closest point from the front to the center point lies on a grid line.

Algorithm *UpdateT* (p)

Input: p , a point in the *UNKNOWN* or *TRIAL* set with at least one neighbor in *KNOWN*.

Output: The new interim arrival time T for p .

(* Initialize or update the interim arrival time for p . *)

1.

(* The set of minimum neighbors in *KNOWN* *)

⁴In general for n dimensions there are 2^n quadrants.

2. $P \leftarrow \emptyset$
3. **for** each 4-neighbor n of p with $n \in KNOWN$
4. $opp \leftarrow$ the opposing neighbor of n
5. **if** $opp \in P$ **and** $T_{opp} < T_n$
6. continue
7. remove opp from P
8. add n to P
9. $T \leftarrow SolveQuadratic(P)$
10. **return** T

The general update procedure for the the arrival time estimate is shown as $UpdateT(p)$. The algorithm selects the neighbors from the $KNOWN$ set under the constraints detailed above. The compiled set of neighbors P is used as input to solve the quadratic equation.

In the concrete case depicted in figure 4.5, we have $P = \{P_0, P_1\}$, so $|P| = n = 2$. Then the arrival time T_c of the front at the center point P_c can be calculated from the arrival times T_0 and T_1 . From the data given in the selected quadrant we can approximate the gradient ∇T_c at the center point P_c using first order accurate finite difference approximations:

$$|\nabla T_c| F(x_c, y_c) = 1 \quad (1)$$

$$\Rightarrow |\nabla T_c| = 1 \quad (2)$$

$$\Rightarrow \left| \left(\frac{T_c - T_0}{\Delta x}, \frac{T_c - T_1}{\Delta y} \right) \right| = 1 \quad (3)$$

$$\Leftrightarrow \sqrt{\left(\frac{T_c - T_0}{\Delta x} \right)^2 + \left(\frac{T_c - T_1}{\Delta y} \right)^2} = 1 \quad (4)$$

$$\Rightarrow \left(\frac{T_c - T_0}{\Delta x} \right)^2 + \left(\frac{T_c - T_1}{\Delta y} \right)^2 = 1 \quad (5)$$

In step 1 to 2 we use $F = 1$ for all points. For equation 3 we approximate the gradient using first order finite differences from the known values T_0 and T_1 in x- and y-direction. To obtain equation 4 we use the definition of the gradient magnitude by euclidean distance. To solve 4 we get rid of the root by squaring to obtain the final quadratic equation 5. With the constraints $T_c > T_0$ and $T_c > T_1$ the quadratic equation can be solved uniquely.

In the general case for a position dependent speed function $F(i, j)$ and the minimal-quadrant points $P = \{P_0, P_1, \dots, P_n\}$ with associated arrival time values T_0, T_1, \dots, T_n , the quadratic equation is

$$\sum_{k=0}^n \left(\frac{T_c - T_k}{\Delta d_k} \right)^2 = \left(\frac{1}{F(i, j)} \right)^2 \quad (6)$$

where Δd_k is the appropriate spatial resolution in the given dimension, for example Δx , Δy , and so on.

Now we give the generic algorithm the ITK⁵ uses to solve the quadratic equation 6 for arbitrary number of dimensions.

Algorithm $SolveQuadratic(P)$

Input: P , the set of neighbors of the point p considered, the arrival time field T . The position dependent speed function F .

⁵National Library of Medicine Insight Segmentation and Registration Toolkit (ITK), <http://www.itk.org/>

Output: The solution to equation 6.

```
(* Solve equation 6 to obtain the new interim arrival time. *)
1.   $x \leftarrow -1$ 
2.   $a \leftarrow 0$ 
3.   $b \leftarrow 0$ 
4.   $c \leftarrow -\frac{1}{F(p)^2}$ 
5.   $P' \leftarrow$  sorted  $P$  so for any  $p_i, p_j \in P', i \leq j: T(p_1) \leq T(p_2)$ 
6.
(* Process  $d$  dimensions. *)
7.  for  $d \leftarrow 0$  to  $|P'|$ 
8.       $n \leftarrow$  the neighbor in  $P'$  at place  $d$ 
9.
(* In case the arrival time calculated is lower than the *)
(* time already attached to a point we consider TRIAL or KNOWN. *)
(* It should only happen for TRIAL points, and as the values are *)
(* ordered with increasing times, we can break out early. *)
10.     if  $x \geq 0$  and  $x < T(n)$ 
11.         break
12.      $a \leftarrow a + 1$ 
13.      $b \leftarrow b + T(n)$ 
14.      $c \leftarrow c + T(n)^2$ 
15.      $dis \leftarrow b^2 - ac$ 
16.      $x \leftarrow \frac{\sqrt{dis} + b}{a}$ 
17. return  $x$ 
```

4.2.3 Heap details

The heap is the central data structure in the Fast Marching Method. It is used to keep an ordered list of interim arrival times, one for each grid point within the TRIAL set. By using a min-heap structure the element with the smallest interim arrival time can be located in $O(1)$ time complexity, while removing and inserting an element into the heap can be achieved in $O(\log N)$ complexity.

For the sake of completeness, we now explain the *min heap algorithm* as used by the Fast Marching Method. We follow the excellent description of heap variants in [3].

A *heap*, is an *almost-complete* binary tree whose elements all fulfill the *heap condition*. Almost-complete means every element except possibly elements in the lowermost layer or possibly rightmost elements in the second lowermost layer have two child nodes. The heap condition constraints the order of nodes in any path from the root node to a leaf node as either non-decreasing (min-heap) or non-increasing (max-heap). For the Fast Marching Method we use a min-heap.

The two procedures that characterize the heap data structure are *SiftUp* and *SiftDown*. Both share the same goal, to restore the *heap condition* after a modification of the heap structure.

Algorithm *SiftUp*

Input: An index i into an array $H[1 \dots n]$ with $1 \leq i \leq n$.

Output: The element at $H[i]$ is moved up and the heap condition is restored.

```

1.  done ← false
2.  if  $i = 1$ 
3.      return
4.  repeat
5.      if  $\text{key}(H[i]) < \text{key}(H[\lfloor i/2 \rfloor])$ 
6.          interchange  $H[i]$  and  $H[\lfloor i/2 \rfloor]$ 
7.      else
8.          done ← true
9.       $i \leftarrow \lfloor i/2 \rfloor$ 
10. until  $i = 1$  or done

```

Algorithm *SiftDown*

Input: An index i into an array $H[1 \dots n]$ with $1 \leq i \leq n$.

Output: The element at $H[i]$ is moved down and the heap condition is restored.

```

1.  done ← false
2.  if  $2i > n$ 
3.      return
4.  repeat
5.       $i \leftarrow 2i$ 
6.      if  $i + 1 \leq n$  and  $\text{key}(H[i + 1]) < \text{key}(H[i])$ 
7.           $i \leftarrow i + 1$ 
8.      if  $\text{key}(H[\lfloor i/2 \rfloor]) > \text{key}(H[i])$ 
9.          interchange  $H[i]$  and  $H[\lfloor i/2 \rfloor]$ 
10.     else
11.         done ← true
12. until  $2i > n$  or done

```

The *Insert* procedure inserts a single element into the heap by appending it to the tree and successively restoring the heap condition using the *SiftUp* procedure.

Algorithm *Insert*

Input: A heap $H[1 \dots n]$ and a new heap element x .

Output: The heap is modified to be $H[1 \dots n + 1]$ and x is element of the heap.

```

1.   $n \leftarrow n + 1$ 
2.   $H[n] \leftarrow x$ 
3.  SiftUp( $H, n$ )

```

The *Delete* procedure removes an element from the heap by swapping it with the last element, shrinking the heap by one element and finally restoring the heap condition again using the *SiftDown* procedure.

Algorithm *Delete*

Input: A nonempty heap $H[1 \dots n]$, and an index i with $1 \leq i \leq n$.

Output: A heap $H[1 \dots n - 1]$ with the element at $H[i]$ removed.

```

1.   $x \leftarrow H[i]$ 
2.   $y \leftarrow H[n]$ 
3.   $n \leftarrow n - 1$ 
4.  if  $i = n - 1$ 

```

```

5.      return
6.   $H[i] \leftarrow y$ 
7.  if  $key(y) \leq key(x)$ 
8.       $SiftUp(H, i)$ 
9.  else
10.      $SiftDown(H, i)$ 

```

The *DeleteMin* procedure removes and returns the topmost element of the heap, the element with the minimum value.

Algorithm *DeleteMin*

Input: A nonempty heap $H[1 \dots n]$.

Output: The element in the heap with the minimum key is returned.

```

1.   $x \leftarrow H[1]$ 
2.   $Delete(H, 1)$ 
3.  return  $x$ 

```

4.3 Implementation

The above algorithms constitute an almost complete implementation of the FMM. However, two things of importance in the context of this thesis have so far been omitted for clarity.

1. Grid coordinate to heap indexing.

In line 6 of the *Fast Marching Method* algorithm the arrival time is updated for all grid points in the 4-neighborhood of a point. As these updated grid points are in the TRIAL set of points, they are kept in the heap structure. Updating – reducing in fact – their arrival time will change their position in the heap. To maintain the heap condition, we first remove the point from the heap and add it again with its updated time.

However, to remove it from the heap we have to know its position within the heap. This is achieved by keeping a “backpointer” function $H(x, y)$ for all grid points in the TRIAL set, that returns the index into the heap. In practice, this is realized by using an extra grid storing the indexes and also exchanging the elements therein upon interchanging them in the heap in both the *SiftUp* and *SiftDown* algorithms.

2. Conversion to a level set.

To convert the FMM arrival time surface T to a levelset, we first decide the arrival time t_a at which we want to extract the contour and then use a standard isocontour extraction on $T'(x, y) = T(x, y) - t_a$. The extracted contour is in turn used to initialize the level set function ϕ .

Chapter 5

Liver Perfusion - Approach

5.1 Overview

A schematic overview of the liver perfusion measurement approach proposed in this chapter is shown in figure 5.1.

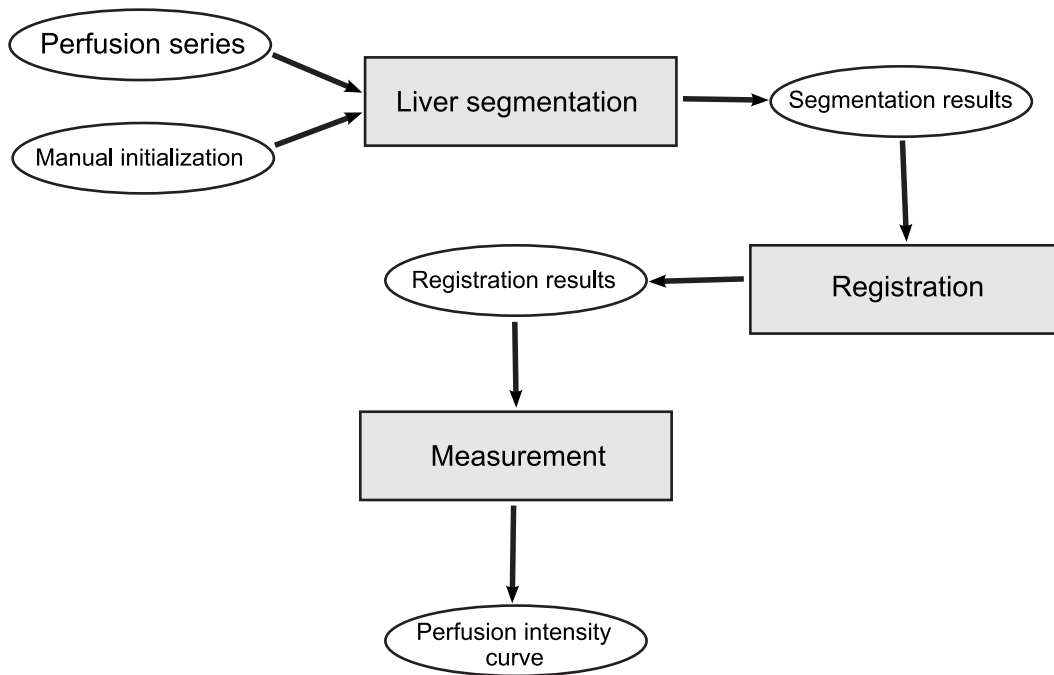


Figure 5.1: Liver perfusion measurement approach: the series and initialization is processed automatically to obtain the perfusion intensity curve.

The initial data provided to the algorithm is the MRI perfusion series set plus initialization parameters. Using a cascade of automatic algorithms the perfusion intensity curve is obtained. The individual steps this chapter is concerned with are the following.

1. **Segmentation**, the process of extracting an approximate shape of the liver from the images.

2. **Registration**, the process of finding the geometric relationship between the extracted liver shapes.
3. **Measurement**, the process of measuring the perfusion intensity across all the images in the series.

The rest of this chapter documents the workings of each step.

5.2 MRI - Magnetic Resonance Imaging

Magnetic resonance imaging (MRI), also called *magnetic resonance tomography* (MRT) is a medical imaging technique based on nuclear magnetic resonance, where nuclei of atoms absorb or emit electromagnetic energy within a magnetic field based on their angular and a magnetic moments. In modern MRI the resonance absorption of the nuclei of hydrogen atoms is used to take slice images of soft biological tissue at arbitrary orientation, resulting in images quite similar to classical x-ray images. X-ray and MRI images are different in a number of respects:

- With MRI one can incrementally obtain different slices of the patient and reconstruct a three dimensional volume from them.
- MRI images have a strong soft tissue contrast, which is used to take images of blood vessels and of the brain, which x-ray is not capable of.
- The spatial resolution of MRI is inferior to x-ray images and the usual pixel or voxel sizes of MRI images are unsuitable to take images of very fine biological structures. A usual good voxel resolution is $8mm^3$ [42]. In comparison, x-ray images offer a very high resolution.

The advantages of MRI have lead to a fast adoption since the 1980s. A good introduction is given by Oppelt [38].

Contrast agents

As image contrast is based on the magnetic properties of the atoms and molecules in the imaged tissue, the chemicals causing a strong contrast are called *contrast agents*. An important natural contrast agent in the human body is *deoxygenated hemoglobin* (Hb), which is paramagnetic. The concentration of Hb varies with the local oxygen supply of the human body and it can be used for *blood oxygen level dependent* (BOLD) contrast in *functional MRI*, as Oppelt explains in [38].

For some clinical applications, artificial contrast agents are used to enhance the image or to study the flow of the contrast agent in blood vessels or organs. In such case, a biologically inert chemical with a strong magnetic response is injected into the patient's body. After the injection, concentration time curves are measured using MRI and by the known properties of the used agent, the concentration agent dependent parameters and physiological parameters are calculated. Prato et al. [42] give a recent introduction into contrast agent modeling.

In the context of this thesis, the contrast agent modeling is relevant to the clinical liver perfusion application, because an artificial contrast agent is used.

5.2.1 MRI data modality

We obtained three perfusion series from the Shanghai First People Hospital. The series consists of two-dimensional 256x256 MRI images taken with a GE Medical Systems Genesis Signa HiSpeed CT/i system at the Shanghai First People Hospital. They show the patients abdomen in coronal view. The imaging parameters are the following: slice thickness 15.0, repetition time 4.7, echo time 1.2, magnetic field strength 15000, flip angle 60 degrees.

The first series consists of 240 images, the second series of 59 images and the third series of 200 images.

5.2.2 Employed data format

The original data format was DCM, the Digital Imaging and Communications in Medicine (DICOM) image format, which is commonly used for the exchange of medical images. The DCM format varies a lot across different vendors, and not every DCM enabled software is able to read this files. I successful read the image data using the opensource tools Medcon¹, CTN² and the ImageMagick toolkit³.

As preprocessing step, the images were first converted to the PNG file format using medcon with the following options: `medcon -e 1 0 -fb-dicom -c png -f *.dcm`. Afterwards the images are converted to Portable GrayMap (PGM) files using ImageMagick. PGM is a simple textual format that is easy to read from and write to.

5.3 Segmenting the liver

We developed a segmentation process for liver shape segmentation employing three steps. The first step locates a seed point for the segmentation in every image. The second step applies the Fast Marching Method to the original image at the given seed point to yield a first approximation of the liver shape. In the third step, this shape is used to initialize a level set segmentation step, which introduces a regularization term to improve the segmentation results and repair local irregularities in the segmented shape. We now describe the steps in detail.

5.3.1 Locating the seed point

The seed point marks the initial curve position in the image. The initial curve is then evolved to segment the shape of the liver. As such, knowing the position of the seed point for every image is the basic requirement to start segmenting it. We use two distinct methods to locate the seed point.

- *Trivial method.*

We ignore any movement and use one initial marked seed point for all images.

- *Gradient maximizing method.*

For one image, the radiologist manually marks the segmentation seed point. For all other images it is located using the following robust, but non-general procedure.

¹Medcon, Medical Image Conversion Utility, <http://xmedcon.sourceforge.net/>

²Central Test Node, a DICOM implementation, <http://www.erl.wustl.edu/DICOM/ctn.html>

³ImageMagick, <http://www.imagemagick.org/>

In almost all series, the top of the liver shape is well contrasted to the background and a strong gradient response is always found at this place. As the patient breathes throughout the series, the liver moves only vertically. Combining this with the strong gradient we get a simple method to reliably obtain a point inside the liver. For every image the gradient magnitude is extracted inside a vertical strip at the horizontal position of the initial seed point along a given expected liver movement window. The values in this strip are mean smoothed. We found a smoothing window of ± 3 good at eliminating local noise. The maximum gradient magnitude is located in the strip, which is at the top boundary of the liver shape. From this, we step a fixed length Δy down to yield a seed point inside the liver shape. The value of Δy is determined once in the original image, for which we already know the seed point location.

Both methods are specific to the liver perfusion problem.

5.3.2 FMM Segmentation Step

The FMM segmentation step takes as input all the images and a single seed point for each image. As result, a segmented liver shape is returned as bitmap of the same dimension as the input image. For each pixel in the map which has a positive truth value the pixel is thought to belong to the liver shape.

The FMM algorithm is fairly straightforward and the only flexible part is the definition of the speed function to use and the stopping criteria. The speed function determines the propagation speed in normal direction for any point in the computational domain. The stopping criteria tells us when to stop the segmentation.

Speed function

The speed function F_{FMM} we use for the FMM segmentation step is a thresholded variation of the well known speed function used in [31]. We first define

$$F_{base}(x, y) = \frac{1.0}{1.0 + |S_k \cdot \nabla(G_\sigma * I_{x,y})|^{S_p}} \quad (1)$$

which is the interface propagation speed in normal direction based on the gradient magnitude image. The gradient image $\nabla(G_\sigma * I_{x,y})$ is the Sobel approximated gradient of the original image I convolved with a Gaussian of width σ . σ , S_k and S_p are constants and we had good results using $\sigma = 3.0$, $S_k = 13.0$ and $S_p = 2.0$ for the series examined.

The speed image is generated from F_{base} by using one of the following two methods.

1. Thresholded F_{base} .

By using F_{base} in the thresholded speed function F_{FMM} , defined as

$$F_{FMM}(x, y) = \begin{cases} F_{base}(x, y) & F_{base}(x, y) \geq S_t \\ 0.0 & F_{base}(x, y) < S_t \end{cases} \quad (2)$$

with S_t being the threshold value. We used a quite high value of $S_t = 0.6$, the reason giving below.

2. Gaussian biased F_{base} .

One natural idea to cut small speeds from F_{base} down even further while not reducing larger speeds is to use a Gaussian function, as shown in equation 3. Here, F_{base} is used in F_{FMM_2} as

$$F_{FMM_2}(x, y) = e^{-\frac{(1-F_{base}(x,y))^2}{2\sigma^2}} \quad (3)$$

Three possible choices for σ , the Gaussian standard deviation, are shown in figure 5.2. A small value leads to a strong cutoff, and almost completely removes small speed values, producing similar results as equation 2. A larger value leads to a smooth reduction towards smaller values.

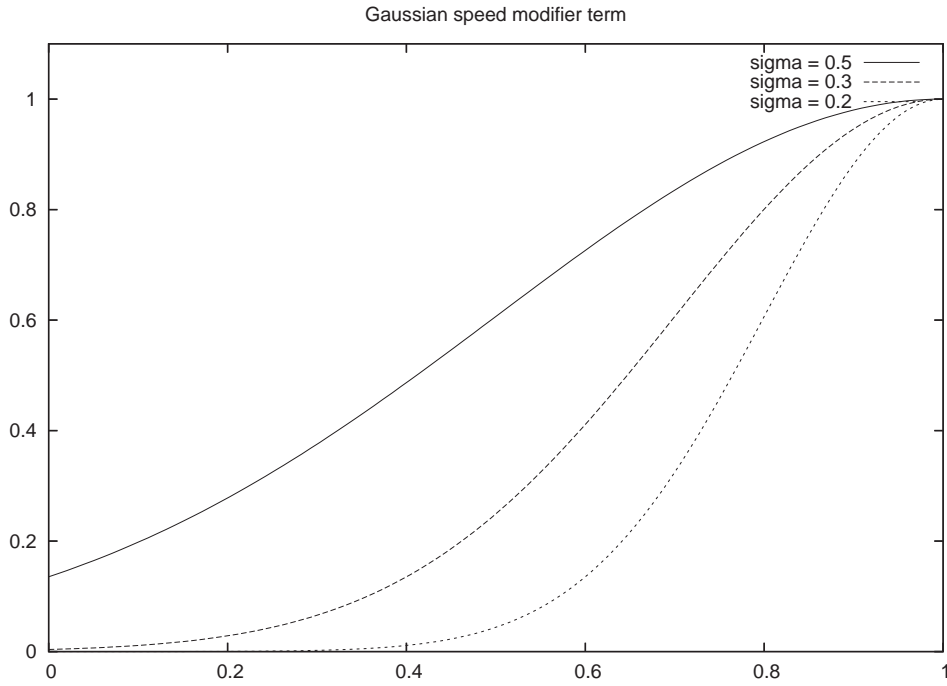


Figure 5.2: Three speed modifier functions based on the Gaussian equation 3, with $\sigma = 0.2$, $\sigma = 0.3$ and $\sigma = 0.5$.

Because the FMM cannot incorporate curvature dependent information, a segmentation using an unthresholded or unbiased speed function such as F_{base} defined in equation 1 could leak out of the liver shape at positions where the gradient is locally weak. By using a cautious threshold or bias we limit the risk of leaking in exchange for a higher probability to not cover the entire liver shape in the first segmentation step. This defect is effectively repaired in the following two segmentation steps, using the more powerful and robust levelset evolution method.

Stopping criteria

The stopping criteria for the FMM segmentation is simply a constant of the area size. As soon as this area size is reached, the segmentation stops. This corresponds to the number



Figure 5.3: Result of the FMM segmentation step.

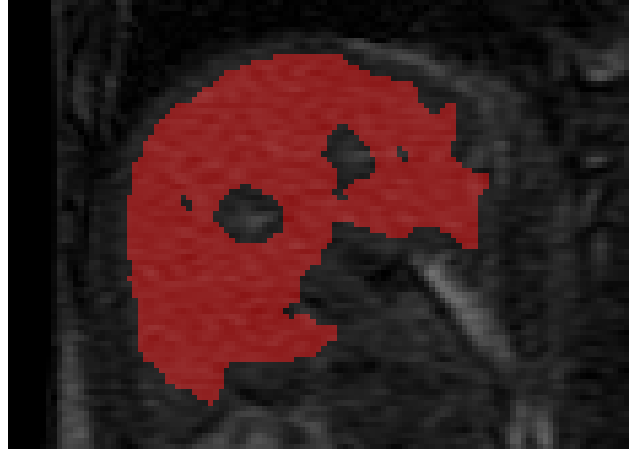


Figure 5.4: Liver region inside image 5.3.

of iterations of the FMM, as it always accepts exactly one new pixel to the shape for each iteration. The value is configurable but the default of 1500 elements yielded good results.

Problems

Consider a typical result of the FMM segmentation step, as shown in figure 5.3 and 5.4. While the overall shape of the liver has been captured to some degree, the shape has the following deficiencies.

1. *Holes.*

The patient's liver shown in figure 5.3 most certainly does not contain any holes. Hence, the captured shape also should not have any. In the FMM segmentation result however, there are four such holes, small and large, which are caused by a strong local gradient at their place. Because of the way F_{base} is defined, the front slows down at strong gradients and holes remain.

2. *Irregular shape.*

The speed function F_{base} depends only on the image and the position of the front inside the image and there is no regularization force acting upon the shape itself. Thus, the resulting shape has many corners, which does not reflect the liver as a smooth shape.

3. *Leaks.*

Leaks occur when the shape's front crosses the real liver contour on the image. There are three distinct reasons for leaks to happen in FMM segmentation.

- (a) $F_{base}(x, y) > 0$ for any (x, y) .

The front is always extending outward in FMM and only the relative ordering of the $F_{base}(x, y)$ values matter, not their absolute size. Hence, there is a time t for any point in the image when the front crosses that point. Given enough time,

leaks are unavoidable. With that in mind, our goal is to construct a good speed function which only leads to leaks once the entire liver shape is already successfully segmented, which is the point we hope to stop the segmentation.

(b) Initialization symmetry.

Surprisingly, FMM segmentation is very sensitive to the segmentation initialization. The reason for this is related to the previous point. As the relative order of the function values $F_{base}(x, y)$ are important, points that are closer to the seed point have a time-advantage of being processed, while points that are more far away from the seed point are reached by the front later in time. Thus, to recover a shape successfully using FMM segmentation, it is important to have the seed in the approximate center of the shape. We try to achieve a good seed position by the method detailed in section 5.3.1.

(c) Missing gradients at object boundary.

The FMM segmentation we employ is agnostic of the real liver shape and can only use information from the image to recover it. It has no understanding of what a liver is supposed to look like. Therefore, two assumptions are important for the segmentation to succeed, (a) that the object of interest, the liver, is delimited by a high gradient response in the image, and (b) that the liver has a regular, smooth shape in the image. In case assumption (a) ceases to hold, the curve leaks over the object boundary and pixels that do not correspond to the liver are taken as being member of the liver shape. Some of the effect a small gradient response causes can be compensated by the regularization force, which we discuss below. In general, it is not possible to avoid leaks due to missing gradients without incorporating assumptions about the spatial expression of the liver shape into the segmentation, which is known as model-driven or atlas-based segmentation and outside the scope of this thesis.

Using a second segmentation step employing level set segmentation, we can improve upon the results and can overcome most of the limitations of the Fast Marching Method segmentation. We now discuss the level set segmentation in detail.

5.3.3 Level set segmentation step

The input to the levelset evolution step is the first rough FMM segmentation result. In this step we incorporate a regularizing curvature term to smooth the shape, remove holes and refine the segmentation result. The result is the level set signed distance map for the entire computational domain.

Speed function

We use a narrow band 2D levelset implementation with a narrow band extending 6 elements into both directions. The speed function used is the original function used above plus a curvature regularizing term. It is

$$F(x, y, t) = \alpha \cdot -\kappa_{x,y,t} + F_{base}(x, y) \quad (4)$$

with $\alpha = 0.4$ constant, $\kappa_{x,y,t}$ being the local curvature at the point (x, y) at time t and the remaining term being the unthresholded gradient based speed term of the FMM

segmentation used above. The negative curvature term removes any small local irregularities the FMM segmentation has left over, such as sharp corners and single non-shape points within the shape due to noise in the original image. Globally it leads to a smoother overall shape.

We use a fixed number of evolution time steps of $\Delta t = 0.2$. To improve the results of the FMM segmentation, we found 80 time steps to be a good value.

Finally, to simplify the perfusion area localization, the narrow band levelset function is extended to a full levelset in the entire domain by redistancing from the shape contour in the narrow band.

Alternative approach

An alternative approach inspired by [54] is to convert the speed function into a convergent one, i.e. one in which for any (x, y) in the domain fulfills

$$\lim_{t \rightarrow \infty} F(x, y, t) = 0.$$

Using the above F_{base} function, we define a new speed function F_c as

$$F_c(x, y) = (F_{base}(x, y) - s) \beta_0 - F_{base}(x, y) \beta_1 \kappa_{x,y,t} \quad (5)$$

with $S_k = 12$, $S_p = 2$ for F_{base} , $s = 0.3$, $\beta_0 = 0.5$ and $\beta_1 = 0.5$ for F_c .

The important part of equation 5 is the ratio $\frac{\beta_1}{\beta_0}$, β_1 being the diffusion term constant and β_0 determining the reaction component. Reaction means the propagation of the contour in normal direction and diffusion means the regularization through the curvature. A good discussion about their relationship is given in [54].

The advantage of equation 5 over equation 4 comes from the ability to be pushed back by strong gradients as a result of the subtrahend s applied to the gradient dependent speed function, while still allowing the full original gradient-dependent speed function to influence the curvature regularization term. This means the reaction part of equation 5 – the constant advancing into the normal direction by speed β_0 – is stable and convergent, while the curvature dependent part controlled by β_1 can overcome any gradient if necessary to regularize the shape. As the shape cannot be regularized ad infinitum, this second term must also converge.

While equation 5 converges, for practical reasons it can also be used to define a stopping criteria, such as

$$S_{F_c}(t) = \left(\sum_{(x,y) \in \Omega} |F_c(x, y)| \right) < S_t, \quad (6)$$

where S_t is a constant defining the absolute change necessary for evolution to continue.

Pushback force

Consider the situation shown in figure 5.5. There two contours surrounding the areas shown in gray. Using both equation 4 and equation 5, the contour surrounding the top area will eventually collapse, as the curvature-dependent forces acting from the outside are quite strong. In case of equation 4 they have no opposing forces but the speed is reduced by a strong gradient. However, when using equation 5 there is an opposing *pushback*-force, which is too small to stop the contour. If we would increase the size of the circle object, the local

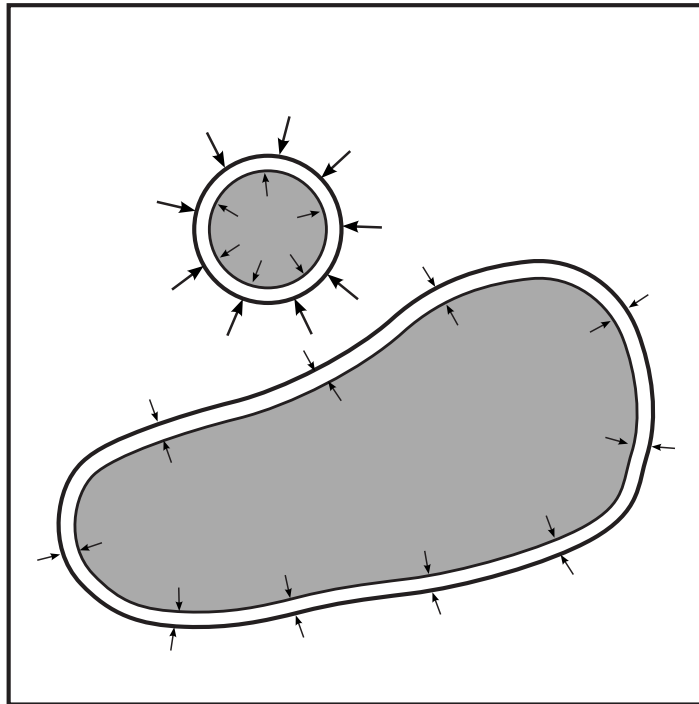


Figure 5.5: Two contours surrounding one object each under influence of the speed function F_c . While both the objects produce the same gradient-based reaction force pushing back the contour, the first contour has a high curvature which overcomes the pushback force (larger forces from outside than from the inside). The second contour's reaction and diffusion forces are in balance and no change happens (equal sized forces/arrows from inside and outside).

curvature decreases and with it the regularization force acting from the outside. At some point, a balance between the pushback- and regularization forces is reached and the contour will not move, which is what happens for the second object, shown at the bottom of figure 5.5. Using equation 5 the forces are in balance and the contour is stable, as shown in the figure. If we would use equation 4, there is no pushback-force and eventually this contour collapses as well.

Other possible speed functions

The success of the segmentation depends on the choice of the speed function. Here we briefly discuss other alternative choices following Droske et al. [12].

Intensity dependent speed functions. Similar to F_{base} , which depends on the gradient image, the following speed functions could be defined depending only on the image.

$$F_{I_1}(x, y) := e^{-\frac{(I_{x,y} - \mu)^2}{2\sigma^2}} \quad (7)$$

Speed function F_{I_1} produces large speed values if the pixel intensity value $I_{x,y}$ in the image is close to a reference gray value μ . The sensitivity is controlled using σ . Using this speed function, a thresholding-like segmentation respecting connectivity can be implemented. However, as the original image intensity $I_{x,y}$ is taken, it is sensitive to noise, and in most cases a better choice would be to first apply a Gaussian smoothing filter to the image.

$$F_{I_2}(x, y) := e^{-\frac{((G_t * I_{x,y}) - \mu)^2}{2\sigma^2}} \quad (8)$$

F_{I_2} does reduce the sensitivity to noise using the Gaussian smoothed image with a standard deviation of t .

Gradient dependent speed functions. Beside the popular function F_{base} we use, these two gradient dependent speed functions are also popular.

$$F_{\nabla_1}(x, y) := e^{-\alpha |\nabla G_\sigma * I|_{x,y}} \quad (9)$$

$$F_{\nabla_2}(x, y) = \frac{1}{1 + \frac{|\nabla G_\sigma * I|_{x,y}^2}{\lambda^2}} \quad (10)$$

In F_{∇_1} , the constant α controls the edge sensitivity. In F_{∇_2} this is done using the constant λ .

Curvature dependent speed functions. In the speed function F , defined as equation 4, we introduce the negative curvature as regularizing term, where the curvature is given as $\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right)$.

This is the standard procedure to introduce a smoothing force, but slight variations exist. For example, to make the normal relative speed function F positive everywhere F_{κ_1} can be used, defined as

$$F_{\kappa_1}(F, x, y, t) = \max(F(x, y) - \epsilon \max(\kappa_{x,y,t}, 0), 0). \quad (11)$$

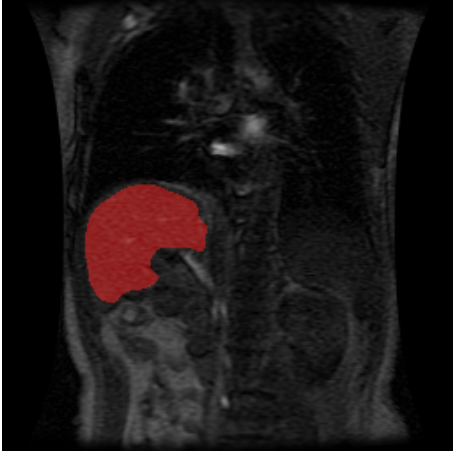


Figure 5.6: Result of the level set segmentation step.

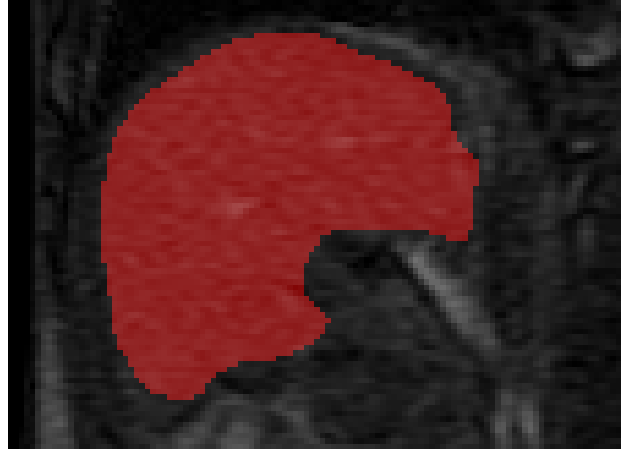


Figure 5.7: Liver region inside image 5.6.

In equation 11, a speed function F is modified so a negative curvature force is introduced. In F_{κ_1} , the curvature force is a limiting force only and cannot push the contour. The overall max term also makes the speed zero or positive everywhere.

This combination of speed functions often makes sense, and for two speed functions F_1 , F_2 commonly the combinations

$$F := F_1 + F_2, \quad F := F_1 \cdot F_2, \quad F := \min(F_1, F_2), \quad F := \max(F_1, F_2)$$

are used.

Levelset segmentation results

A typical result of the level set segmentation step is shown in figure 5.6, with an enlarged version in figure 5.7. Compared with the FMM segmentation results in figure 5.3, the shape is much more regular, hole free and captures the liver better. At the bottom part of the liver there is a small leak due to the low absolute gradient response there. In general, two of the three problems mentioned above for the FMM segmentation, the shape irregularity and the holes can be solved using level set segmentation. The third problem, leaking of the shape over the object boundary, can be reduced by the regularization term, but not completely solved.

Why can our level set segmentation not overcome the leaking problem? The speed function we use for the segmentation uses two assumptions to recover the liver shape: (a) the liver shape is regular, smooth in the image, and (b) the liver boundary has a gradient response associated with it. If one of them is locally not true, the shape may not be recovered successfully. In the image shown in figure 5.7, the gradient response is small on the complete bottom side of the liver, hence the contour leaks over it. If it would be locally small, such as a small gap on an otherwise strong line in the gradient image, the regularization force can prevent leaking of the contour. In our case, this is not possible and the only thing that can be done without modifying the assumptions is to stop segmentation early enough. More generally, in order to overcome the leaking problem we have to incorporate model driven term into

the segmentation that can guide the contour and stop leaking even when a small or even no gradient response is present.

The results of the level set segmentation step are used to locate the perfusion area in each image, the exact details of which we describe now.

5.4 Perfusion Area Localization

After the segmentation of the liver area is completed for all images, we locate the area where to measure the perfusion intensity in every image.

We now introduce a new method to locate this area. In this method, the radiologist first marks the perfusion area in one image. Afterwards, this area is automatically located in all remaining images.

Our method is based on the following five observations. First, the movement of the liver is constrained to mainly vertical movements with only small horizontal movement. Second, shape of the segmented liver changes only slightly throughout the series. Third, rotation relative to the body is minimal and can be ignored. Fourth, the segmentation quality is not equal across the entire shape but the best at the top half of the liver, due to a strong gradient response there, while the weak gradient response in the lower half of the liver leads to more variation throughout the series. Fifth, when the contrast agent reaches the perfusion area a strong gradient response appears, which changes the segmentation result locally, up to the case of where the perfusion area is not considered part of the liver shape anymore.

5.4.1 Distance Vector Transform

We now describe the details of the method. The idea is to anchor a coordinate system at each segmented shape which can be used to estimate a point location within and nearby the liver shape across all the images. As the method employs limited redundancy it also allows for a definition of an error term which can be used to evaluate the relative quality of the localization among the images.

Definition

Consider the point P in figure 5.8. Assume for now, we know for sure the position of P absolutely in the image and that we have a good segmentation result of the liver shape. Then, we define a set of non-parallel lines $L = \{L_0, L_1, \dots, L_n\}$. For each line L_k we determine the following for the image I

1. The distance $d(P_{x,y}, L_k, I)$ of every point $P_{x,y}$ within the segmented liver shape to L_k .
2. The shortest distance $d_s(L_k, I)$ among all $d(P_{x,y}, L_k, I)$.

Then, any point within the liver in an image I can be represented as a *line relative distance vector* $D_{P_{x,y}}(I)$:

$$D_{P_{x,y}}(I) := (d(P_{x,y}, L_0, I) - d_s(L_0, I), d(P_{x,y}, L_1, I) - d_s(L_1, I), \dots, d(P_{x,y}, L_n, I) - d_s(L_n, I))$$

For example, for the trivial case let $L = \{L_0, L_1\}$ with L_0 being a vertical line through the origin, and L_1 being a horizontal line through the origin. The resulting $D_{P_{x,y}}$ is nothing

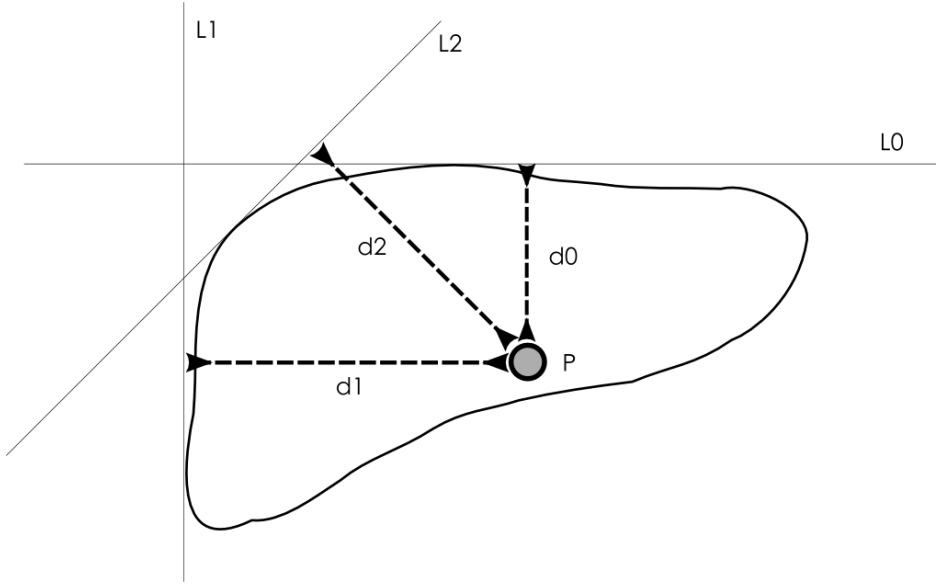


Figure 5.8: A point P defined by a three element distance vector (d_0, d_1, d_2) relative to the lines L_0 , L_1 and L_2

but (x_r, y_r) , the relative coordinates of the point $P_{x,y}$ to the leftmost point and the topmost point of the liver shape.

Using the above model, what remains to be discussed is how to find suitable lines to build the distance vector from and - equally important - how to transfer from a given distance vector and the lines back to an absolute coordinate. The question of which lines to use can be answered by considering the fourth observation made above. We use lines which always have their minimum distance point at the boundary of the shape where the segmentation result is of good quality. For example, due to the strong gradient at the top of the liver, the segmentation result is always good there. Then, by using a horizontal line above the liver as distance measurement line, the resulting component in the distance vector will accurately reflect the relative position to the top of the liver across all the images. Similarly, the top left part of the liver is always well segmented and by fitting a diagonal line with a 45 degree angle, we yield another good element for the distance vector. We now discuss in detail the second question, how to convert a distance vector back to absolute coordinates.

Given $D_P(I_p)$ for the initial radiologist-marked image I_p and the segmentation results for all other slices, we determine the position within or nearby the segmented shape that minimizes an error term.

Error term

We first describe this error term and then show how it is used to find the error minimizing point in the image. Consider a two dimensional coordinate system. For such a system, two non-parallel lines are enough to define a base and any additional lines are redundant. This redundancy can be used to define an error term ε which describes how much the distance vector for a point (x_0, y_0) in image I_0 diverges geometrically from the distance vector of the

original known perfusion area point (x_p, y_p) in the radiologist-marked image I_p :

$$\varepsilon(x_0, y_0, I_0) := \left| D_{P_{x_0, y_0}}(I_0) - D_{P_{x_p, y_p}}(I_p) \right|$$

The *error minimizing point* (x, y) in the image I is one of the points that minimizes $\varepsilon(x, y, I)$. To find this point we test all points (x, y) for which the level set value is below some small positive threshold value t and keep the minimal error point. Because the embedding function used with the levelset method is the signed distance function, the use of a threshold of zero would correspond to all points within the segment liver. Adding a small positive value t allows for points nearby the segmented shape to be found and is identical to a number of dilation operations using a circular disc. The threshold value is necessary because of the observation made above about the strong gradient response when the contrast agent reaches the liver, where the perfusion area may lie slightly outside of the (bad) segmented shape.

After all the minimizing error points are located in the images, the perfusion area is simply a circle area centered around the point in each image. Because the liver shape is deformed only slightly throughout the series and the circle area is invariant to rotation, it is a simple but sufficient approximation.

For each image, the resulting perfusion intensity is the mean value of all the MRI intensity values within the circle area⁴. As final step, the resulting curve of perfusion intensity over time is Gaussian-smoothed using $\sigma = 3.0$ to remove variation from intensity values resulting from outliers.

⁴The unit of this intensity is arbitrary and to obtain the contrast agent concentration from them requires more effort and also depends on which contrast agent is used.

Chapter 6

Liver Perfusion - Experiments and Results

In this chapter we apply the approach to real world perfusion series and evaluate the results. First, I give details about the prototype implementation program that realizes the approach and describe the typical user behavior. Secondly, the experiment is described, and third the results are evaluated and discussed.

6.1 Prototype implementation

The prototype implementation has the following features.

- 2D implementation of the narrow band level set method and the fast marching method written in C.

All algorithms described in chapters 3 to 5 have been implemented in C. The code has been optimized for speed but does not make use of any adaptive data structure, such as a quad-tree to organize the computational domain¹.

- Reading and writing PGM² image files.

The relatively complex DICOM file format has not been implemented but instead a simple common text based format (PGM) is used to load and save images from the program. DICOM files can be converted to PGM files using tools such as medcon and ImageMagick.

- A wrapper to a high level C# interface.

To have a good compromise between performance and rapid development speed, the performance critical functions have been developed entirely in C, but for the GUI and its interactions, the higher level C# language has been used. The necessary interfaces to interoperate between the C and C# code have been semi-automatically generated using SWIG³.

¹Extension to 3D for large domains would require using such an adaptive structure to avoid extensive memory consumption. One such approach can be found in [12].

²PGM, Portable GrayMap format from the ImageMagick tools.

³SWIG, Simplified Wrapper and Interface Generator, <http://www.swig.org/>

- Gtk# based GUI written in C#.

Using the portable Gtk# GUI toolkit⁴ only one source is used to compile binaries for Windows as well as Linux; enabling a platform independent development. The perfusion intensity plot has been realized using the NPlot library⁵.

6.1.1 Graphical User Interface

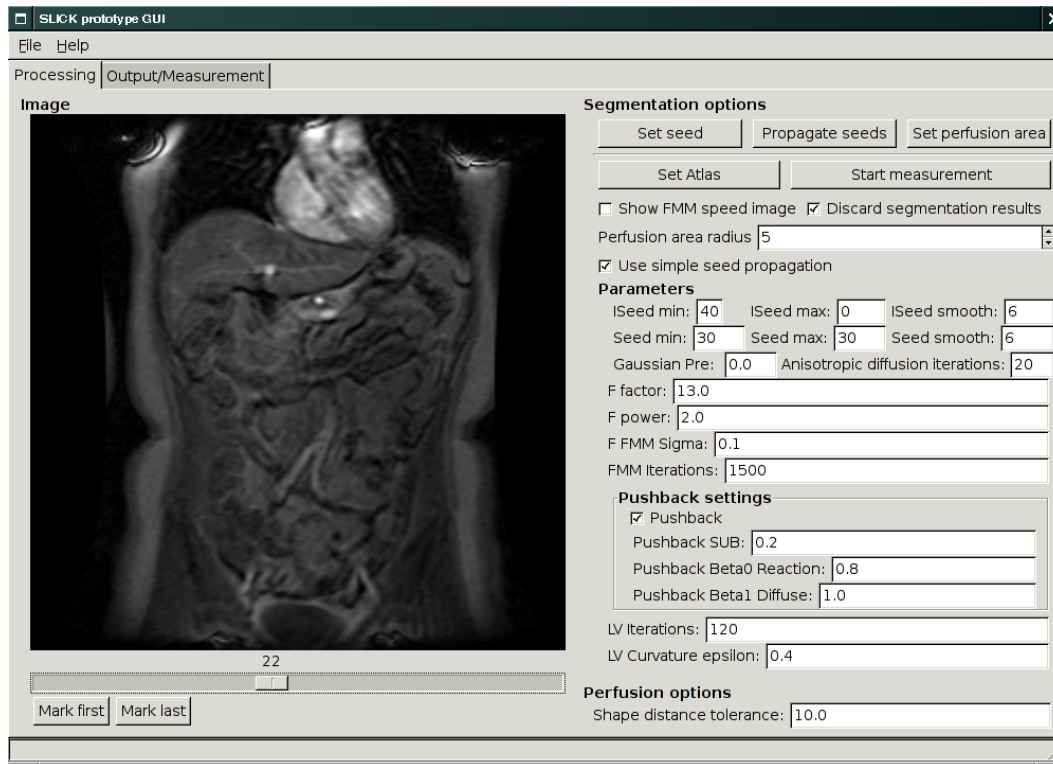


Figure 6.1: Graphical user interface of the prototype implementation in the initialization and configuration view.

The screen shown in figure 6.1 shows the main interaction point between the user and the program. On the left side the MR images are shown for examination and selection. All the loaded images can be selected using the slider below the image; it is also possible to use the cursor keys to browse between the images. On top of the image three elements can be displayed, namely (a) the seed point used to start the segmentation of the image (red tinted circle), (b) the approximate segmented shape of the liver (yellow area) and (c) the located perfusion area (green tinted circle).

On the right side in figure 6.1 the possible actions and configuration settings are shown. The four available actions are (a) set seed point, (b) propagate seed, (c) set perfusion area and (d) start measurement. The normal way for a user to interact with the program is the following:

⁴Gtk#, .NET binding to GTK, <http://gtk-sharp.sourceforge.net/>

⁵NPlot, <http://netcontrols.org/nplot/>

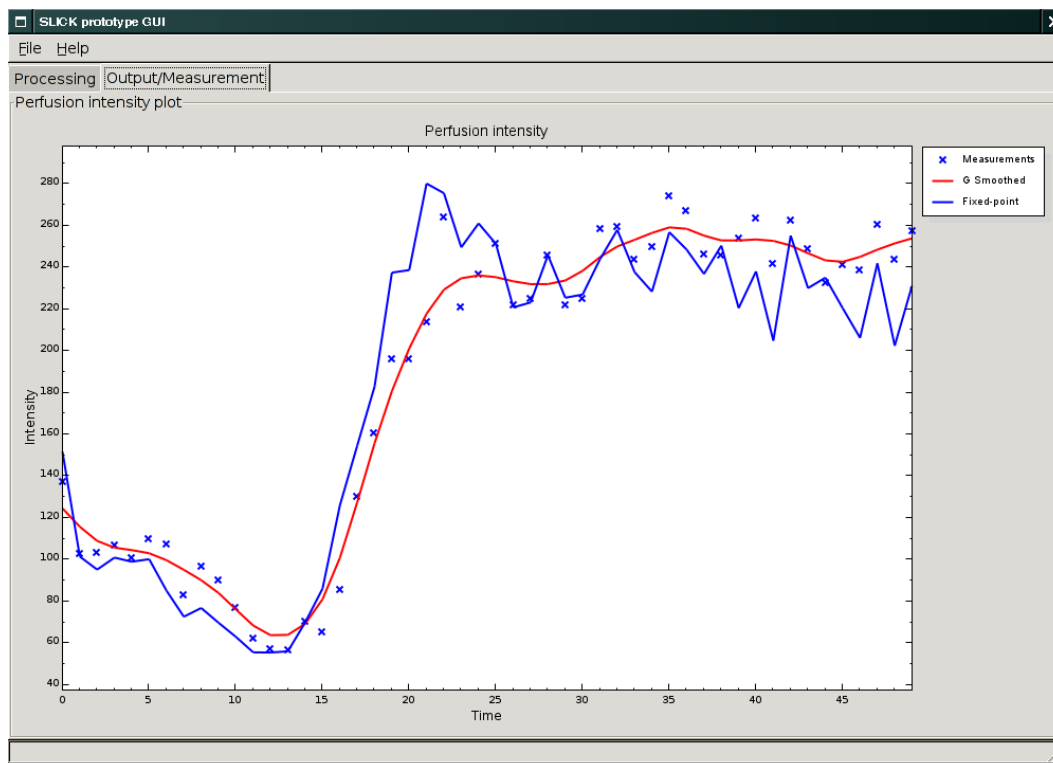


Figure 6.2: Graphical user interface of the prototype implementation in the perfusion intensity measurement view.

1. Load the images through the **File/Open** menu item.
2. In one image: set a seed point by pressing the “**Set seed point**” button and subsequently clicking into the image.
3. Propagate the seed to all other images by pressing the “**Propagate seeds**” button.
4. Selecting the perfusion area. For this the user first selects one image in which the perfusion area is particularly salient and subsequently presses the “**Set perfusion area**” button. The following click in the image sets the perfusion area. This has to be done for one image only.
5. Starting the segmentation and measurement process by pressing the “**Start measuring**” button.

In figure 6.2 you see the resulting perfusion curve as displayed in the program. The plotted graph displays both the measured values (blue crosses) and a Gaussian smoothed curve (red line).

6.2 Experiments

The experiments are the following.

1. Clinical evaluation.

The measured perfusion curves have been discussed with radiologists from the Shanghai First People’s Hospital and the general accuracy has been discussed.

2. Performance evaluation.

The runtime performance is taken for a common system configuration.

3. Segmentation accuracy evaluation.

Ultimately, the location accuracy of the perfusion area is the most important, as the intensity measurement is taken from it. However, the segmentation accuracy is important as well, because its results are used by the registration algorithm to locate the perfusion area. Hence I use the following two-step method of testing the segmentation for accuracy: (a) the top of the liver is manually marked in a large representative set of images, and (b) the vertical position is compared with the topmost position in automatically segmented liver shapes for these images. The difference between the two measurements will give information about the segmentation performance.

The choice of only comparing points is based on the registration method used. The segmentation quality is not equal throughout the liver shape, hence we take the actual positions used by the registration method instead of relying on potentially bad parts of the segmentation.

4. Perfusion area location accuracy evaluation.

The perfusion area location accuracy can be evaluated by the error term produced in the registration step. I list the error terms for all series and interpret them geometrically.

6.2.1 Parameters

For the evaluation, all freely configurable settings in the program are set as shown in figure 6.1. The intrinsic parameters were the following. We use F_{FMM_2} for the FMM segmentation step. The set L of distance measurement lines were a horizontal and vertical line at the origin plus a 45 degree line in mathematical positive rotation through the coordinate system origin.

The evaluation has been performed using the prototype GUI in the manner described in section 6.1.1. One perfusion series is loaded and a particular good image of the series is selected and both the segmentation seed point and the perfusion area is manually marked in this image. Then the seed point is automatically located in all the remaining images and can be examined. Afterwards the actual segmentation and perfusion measurement process is started. The output of the measurement process is the intensity curve at the estimated perfusion area and the minimal error term value at that position for each slice.

6.3 Results and discussion

The results and interpretation to the proposed experiments are given here.

6.3.1 Clinical evaluation

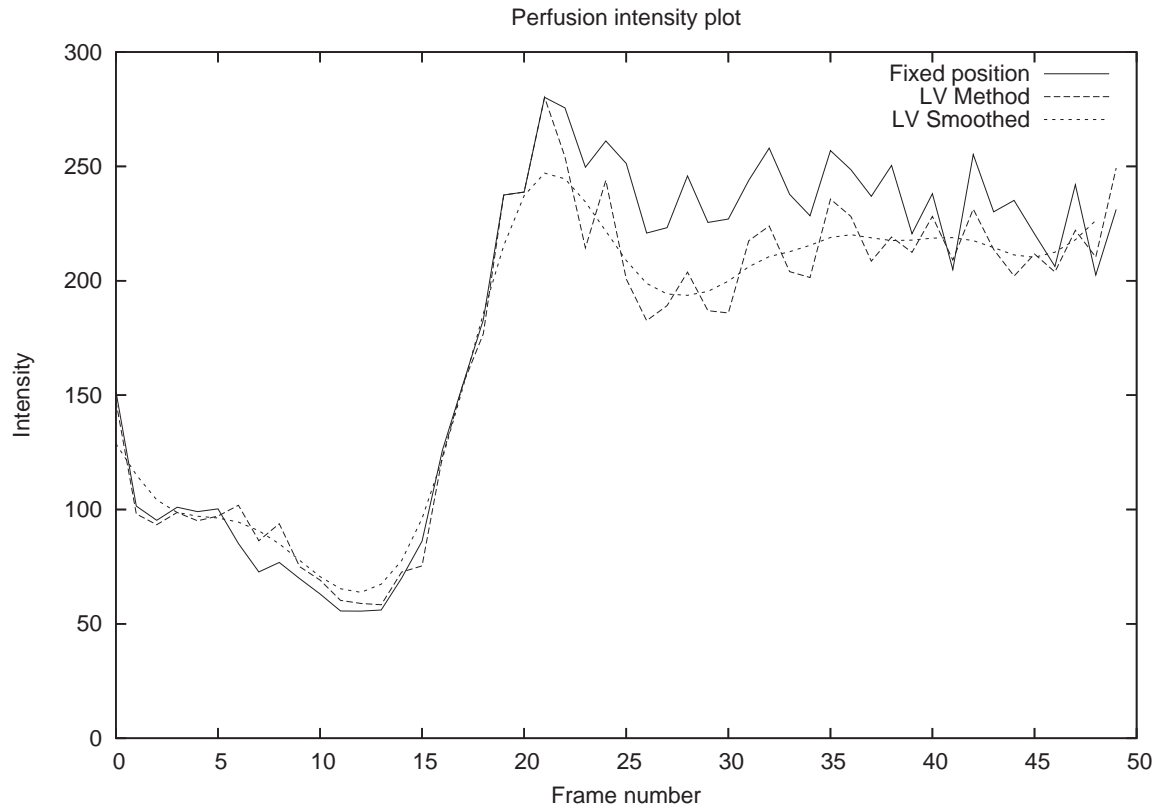


Figure 6.3: Typical liver perfusion intensity curve obtained with the prototype. Shown for comparison are the results of the original method using a fixed point across the series, the measured results using our method and the smoothed final results of our method.

In figure 6.3 the Gaussian smoothed measured perfusion intensity is plotted over time. Clearly visible around $t = 15$ to $t = 20$ is the drastically increased intensity as the contrast agent reaches the perfusion area.

Two radiologists from the Shanghai First People's Hospital confirmed that the results produced using our method are an improvement over their current method. The reasons for the improvement are twofold:

1. Compensation of movement, that is, the movement of the liver is compensated for.
2. Filtering of the result curve. While only the compensation of movement already leads to an improvement, both the original data and our compensated data benefit from filtering it through an interpolation or approximation filter. In figure 6.3 we used a Gaussian filter of $\sigma = 1.5$, but we obtained similar results using BC-spline filtering.

6.3.2 Performance evaluation

For the performance evaluation, a combined set of 240 images has been used. The system is a Pentium-M 1500Mhz, 512Mb RAM system running Linux 2.6.6 with the Mono 1.0.5 CLR.

Step	time	time per image
Locating seed point	44s	0.183s
FMM and levelset liver segmentation	443s	1.845s
Locating perfusion area	23s	0.095s
Total	510s	2.125s

The absolute runtime performance of around two seconds per image is adequate and compares favorable to manual marking of the perfusion area in every image, while being slower than an instant simple non-compensating measurement.

6.3.3 Segmentation accuracy evaluation

The overall segmentation accuracy is not too good, and the leaking problem is not easy to solve. Of importance for the registration is only the part of the segmentation result that is used as input to the registration. One such part is the top point of the liver, the point inside the segmented shape that has the smallest vertical coordinate.

In figure 6.4 the vertical coordinate top point of the liver is shown over 121 pictures of two combined perfusion series⁶. Additionally manually marked positions are shown. Besides being important for the registration, the top point is also easy to find manually and the manual results are believed to be correct. The low mean of the error term shows the segmentation at the top of the liver to be quite good.

6.3.4 Perfusion area location accuracy evaluation

In table 6.3.4 we analyze the error terms within twelve of the thirteen series⁷. Interpreting the error values as the geometric distance from an optimal fit, the low mean error values for

⁶The images of the two series are joined in sequence, so that all images from the second series follow the one from the first.

⁷Series number 11 failed to load into our prototype program for unknown reasons.

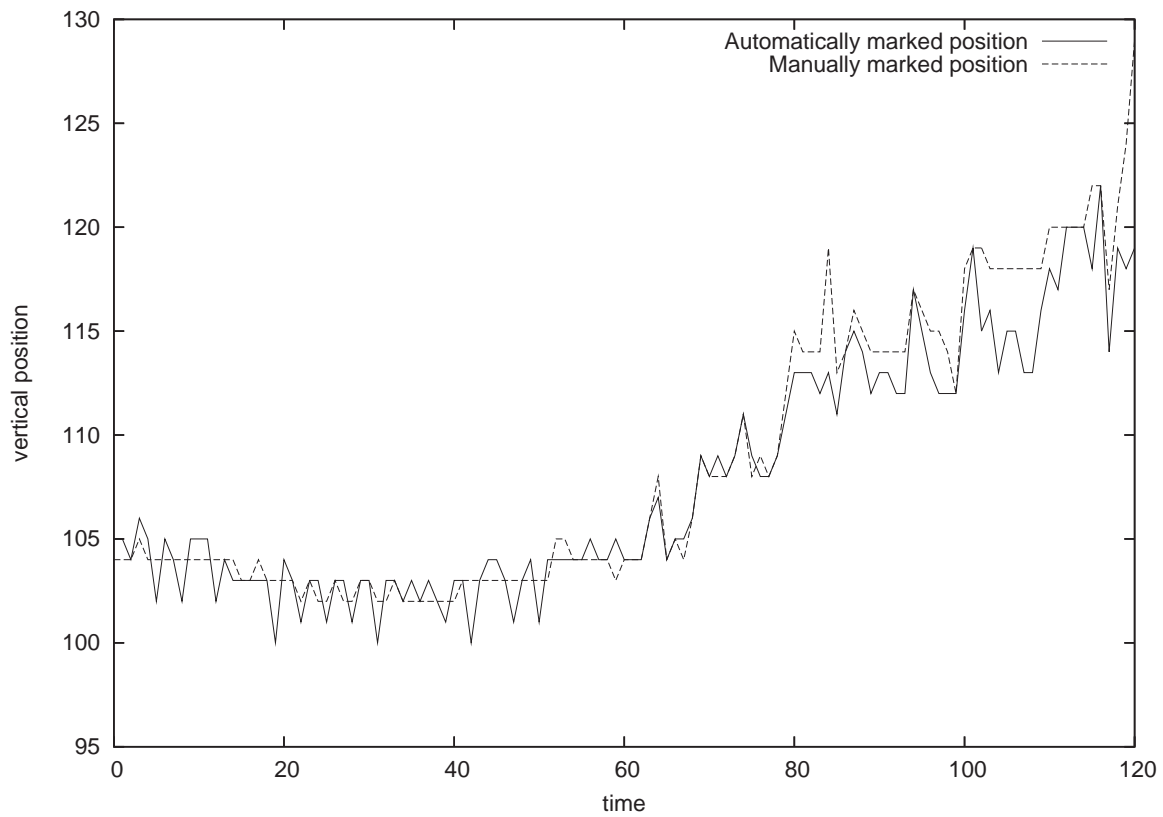


Figure 6.4: Comparing the automated results of finding the vertical pixel position of the top boundary of the liver against manual marking for 121 pictures in two combined perfusion series. The error function $\epsilon(t) = |\text{manual}(t) - \text{automatic}(t)|$ has a mean of 1.2149 pixels with a standard deviation of 1.5395 pixels and a maximum value of 10 pixels.

Series	Error mean	Error maximum	Error standard deviation
1	0.94	3.53	0.78
2	1.03	3.53	0.71
3	3.00	6.00	2.08
4	3.02	5.05	1.06
5	1.64	5.05	1.36
6	1.26	4.53	1.01
7	1.44	3.08	0.84
8	2.86	13.34	2.13
9	1.86	3.54	1.06
10	1.79	5.05	1.25
12	4.44	11.02	3.10
13	0.98	3.08	0.71

Table 6.1: Error term comparisons for the twelve of the thirteen example perfusion series.

all the series - except for series 8 and 12 - indicates a good perfusion area localization success. This is confirmed by manually inspecting the located perfusion area.

For series 8 and 12 the segmentation has failed to accurately recover the liver shape and the following registration step lead to a mislocated perfusion area.

Chapter 7

Conclusions

The following things have been achieved by this thesis's work.

- The main difficulties in the liver perfusion problem have been identified.
- A robust, automatic and performant segmentation and measurement process has been devised for liver perfusion MRI series.
- The level set method, the narrow band level set method and the Fast Marching Method have been implemented, tested and optimized for its application in the liver perfusion approach.
- Different aspects of the approach have been evaluated, (a) the overall approach against its value in clinical applications, (b) the prototype's runtime performance on a common system configuration, (c) the segmentation accuracy and (d) the perfusion area location accuracy.
- Throughout the thesis I gave brief explanations for possible alternative approaches to the segmentation problem. Together they give rise to the possible directions of future work described below.

The problem introduced in section 1.1 has been solved and the goals introduced in section 1.2 have been achieved. The clinical value has been validated by radiologists. While the overall approach as such is a valuable one, there are still deficiencies we hope to remove in the future. I now give final remarks on these possible future developments.

7.1 Future work

The possible future work can be divided into two parts, namely (a) improving the approach of this thesis to obtain better results, and (b) adapting the approach to related fields.

Improving the approach

We have seen the segmentation results have not been optimal. While a part of this problem is caused by the bad input image quality, the larger part is caused by the segmentation algorithm not knowing anything in advance about the shape to segment. Thus, by incorporating a prototype model of the liver and letting it fit to and guide the segmentation of the liver, the results will improve. This is known as Atlas-based segmentation.

Adapting the approach

The current approach manages to measure the perfusion intensity at an area inside the liver for 2D time series. However, the notion of perfusion occurs in other organs as well. A prime example is the *heart*; measuring the perfusion intensity throughout a time series can be of high clinical value and thus an extension of this approach to the heart could be pursued.

The approach could also be extended to 3D images. Usually, extending algorithms from 2D to 3D has two disadvantages, namely (a) the added computational complexity, and (b) introducing new algorithmic complexity to cope with the extra dimension. In the approach this thesis describes both problems would be minimal. The added computational complexity could be reduced by using an adaptive approach to the grid discretization. The added algorithmic complexity, if any, would be minimal, as both the level set method and the Fast Marching Method extend to 3D easily. A large part of the benefit of extending the approach to 3D would be added accuracy in the measurement. Because a natural image of the human body is 3D, the 2D slices we use are always just an approximation of a part of the 3D volume it is contained in. In this 2D slice it is difficult to interpret certain structures and to relate them to be either caused by noise or being an artifact of interpreting 3D data as 2D slice at that place¹.

¹For example, imagine a small bright spot on a 2D slice. This spot could be just noise and in that case should be ignored by the segmentation process. It could also be a part of a thin anatomical structure running perpendicular to the image plane. In that case, its structure cannot be captured by the 2D image alone, but ignoring the structure discards possibly important information from the image. The level set segmentation step we employ with a negative curvature flow will remove such small structures from the image, discarding its information. Incorporating – making sense of it in fact – this information by allowing segmentation to happen in 3D would improve the approach.

Bibliography

- [1] D. ADALSTEINSSON AND J. SETHIAN, *A fast level set method for propagating interfaces*, 1995.
- [2] M. AKAY, A. MARSH, ET AL., *Information Technologies in Medicine, Volume I: Medical Simulation and Education.*, John Wiley and Sons, 2001. ISBN 0-471-38863-7.
- [3] M. ALSUWAIYEL, *Algorithms Design Techniques and Analysis*, World Scientific Publishing Co. Pte. Ltd., 1998. ISBN 9-810-23740-5.
- [4] C. BAILLARD ET AL., *Robust adaptive segmentation of 3d medical images with level sets*.
- [5] M. BARTSCH, T. WEILAND, AND M. WITTING, *Generation of 3d isosurfaces by means of the marching cube algorithm*, IEEE Transactions on Magnetism, 32 (1996), pp. 1469–1472.
- [6] S. BEUCHER, *Watersheds of functions and picture segmentation*, Proceedings of the IEEE International Conference on ICASSP 82 on Acoustics, Speech and Signal Processing, 7 (1982), pp. 1928–1931.
- [7] K. R. CASTLEMAN, *Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 1996. ISBN 0-132-11467-4.
- [8] T. CHAN AND L. VESE, *Image segmentation using level sets and the piecewise-constant mumford-shah model*, 2000.
- [9] S. CHEN, B. MERRIMAN, S. OSHER, AND P. SMEREKA, *A simple level set method for solving stefan problems*, J. Comput. Phys., 135 (1997), pp. 8–29.
- [10] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms, 2nd Ed.*, The MIT Press, 2001. ISBN 0-262-03293-7.
- [11] G. H. COTTET, J. DEMONGEOT, M. E. AYYADI, AND F. LEITNER, *Image segmentation using snake-splines and anisotropic diffusion operators*, July 1995.
- [12] M. DROSKE, B. MEYER, M. RUMPF, AND C. SCHALLER, *An adaptive level set method for medical image segmentation*, Lecture Notes in Computer Science, 2082 (2001), pp. 416–??
- [13] R. DUDA, P. HART, AND D. STORK, *Pattern Classification*, John Wiley and Sons, 2001. ISBN 0-471-05669-3.

- [14] J. S. DUNCAN, L. H. STAIB, S. SCHULTZ, AND X. ZENG, *Volumetric layer segmentation using coupled surfaces propagation*, July 1998.
- [15] R. FISHER, S. PERKINS, A. WALKER, AND E. WOLFART, *Hypermedia Image Processing Reference (HIPR)*, John Wiley and Sons, 1997. ISBN 0-471-96243-0.
- [16] T. A. GARRITY, *All the Mathematics You Missed : But Need to Know for Graduate School*, Cambridge University Press, 2001. ISBN 0-521-79707-1.
- [17] R. C. GONZALEZ AND P. WINTZ, *Digital Image Processing (2nd Ed.)*, Addison-Wesley, Reading, MA, 1987. ISBN 0-201-11026-1.
- [18] R. C. GONZALEZ AND R. E. WOODS, *Digital image processing*, Prentice hall, Upper Saddle River, New Jersey, 2nd ed., 2001.
- [19] S. R. GUNN AND M. S. NIXON, *A robust snake implementation; a dual active contour*, IEEE Trans. Pattern Anal. Mach. Intell, 19 (1997), pp. 63–68.
- [20] A. HARTEN, B. ENGQUIST, S. OSHER, AND S. CHAKRAVARTHY, *Uniformly high-order accurate essentially non-oscillatory schemes*, J. Comput. Phys., 71 (1987), pp. 231–303.
- [21] M. T. HEATH, *Scientific Computing: An Introductory Survey*, McGraw-Hill, New York, NY, USA, 1997. ISBN 0-07-027684-6.
- [22] S. HO, E. BULLITT, AND G. GERIG, *Level set evolution with region competition: Automatic 3-D segmentation of brain tumors*, Tech. Rep. TR01-036, Nov. 2001.
- [23] L. JI AND H. YAN, *Loop-free snakes for image segmentation*, in Proceedings of the 1999 International Conference on Image Processing (ICIP-99), Los Alamitos, CA, Oct. 24–28 1999, IEEE, pp. 193–197.
- [24] M. KASS, A. WITKIN, AND D. TERZOPOULOS, *Snakes: Active contour models*, Academic Publishers, (1987).
- [25] R. KECK, *Reinitialization for level set methods*, 1998.
- [26] B. B. KIMIA AND K. SIDDIQI, *Parts of visual form: Computational aspects*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 17 (1995), pp. 239–251.
- [27] M. LEVENTON, W. GRIMSON, AND O. FAUGERAS, *Statistical shape influence in geodesic active contours*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-00), Los Alamitos, June 13–15 2000, IEEE, pp. 316–323.
- [28] N. LIN, W. YU, AND J. S. DUNCAN, *Combinative multi-scale level set framework for echocardiographic image segmentation*, Lecture Notes in Computer Science, 2488 (2002), pp. 682–??
- [29] A. MAGID, S. ROTMAN, AND A. WEISS, *Comments on “picture thresholding using an iterative selection method”*, IEEE Transactions on Systems, Man and Cybernetics, 20 (1990), pp. 1238–1239.

- [30] R. MALLADI, R. KIMMEL, D. ADALSTEINSSON, G. SAPIRO, V. CASELLES, AND J. SETHIAN, *A geometric approach to segmentation and analysis of 3d medical images*, 1996.
- [31] R. MALLADI AND J. A. SETHIAN, *Image processing via level set curvature flow*, Proc. Natl. Acad. of Sci., 92 (1995), pp. 7046–7050.
- [32] ———, *An $O(N \log N)$ algorithm for shape modeling*, Proceedings of the National Academy of Sciences, 93 (1996), pp. 9389–9392.
- [33] T. MCINERNEY AND D. TERZOPOULOS, *Topologically adaptable snakes*, in Proc. Of the Fifth Int. Conf. On Computer Vision (ICCV'95), Cambridge, MA, USA, June 1995, pp. 840–845.
- [34] ———, *T-snakes: Topology adaptive snakes*, Medical Image Analysis, 4 (2000), pp. 73–91.
- [35] A. MEYER, *A linear time oslo algorithm*, ACM Transactions on Graphics, 10 (1991), pp. 312–318.
- [36] F. MEYER AND S. BEUCHER, *Morphological segmentation*, Journal of Visual Communication and Image Representation, 1 (1990), pp. 21–46.
- [37] C. MONTANI, R. SCATENI, AND M. SCOPIGNO, *Discretized marching cubes*, in Proceedings of the IEEE Conference on Visualization 1994, IEEE, Oct. 1994, pp. 281–287.
- [38] A. OPPELT, *Magnetic resonance tomography - imaging with a nonlinear system*, 2001.
- [39] S. OSHER AND R. FEDKIW, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, 2003. ISBN 0-387-95482-1.
- [40] S. OSHER AND J. A. SETHIAN, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, Journal of Computational Physics, 79 (1988), pp. 12–49.
- [41] N. OTSU, *A threshold selection method from gray level histograms*, IEEE Trans. Systems, Man and Cybernetics, 9 (1979), pp. 62–66. minimize intra and inter class variance.
- [42] F. S. PRATO, C. A. MCKENZIE, R. E. THORNHILL, AND G. R. MORAN, *Functional imaging of tissues by kinetic modeling of contrast agents in mri*, 2001.
- [43] T. W. RIDLER AND S. CALVARD, *Picture thresholding using an iterative selection method*, IEEE Transactions on Systems, Man and Cybernetics, SMC-8 (1978), pp. 630–632.
- [44] G. SAKAS, G. KARANGELIS, AND A. POMMERT, *Advanced applications of volume visualization methods in medicine*, 2001.
- [45] H. SAMET, *The quadtree and related hierarchical data structures*, ACM Computing Surveys, 16 (1984), pp. 187–260.
- [46] J. SETHIAN, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry*, Cambridge University Press, 1998. ISBN 0-521-64557-3.

-
- [47] J. A. SETHIAN AND R. MALLADI, *A real-time algorithm for medical shape recovery*, 1998.
 - [48] C. SHU AND S. OSHER, *Efficient implementation of essentially non-oscillatory shock capturing schemes*, J. Comput. Phys., 77 (1988), pp. 439–471.
 - [49] ———, *Efficient implementation of essentially non-oscillatory shock capturing schemes II (two)*, J. Comput. Phys., 83 (1989), pp. 32–78.
 - [50] K. SIDDIQI, B. B. KIMIA, AND C.-W. SHU, *Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution*, Graphical models and image processing: GMIP, 59 (1997), pp. 278–301.
 - [51] M. ŠONKA, V. HLAVÁČ, AND R. D. BOYLE, *Image Processing, Analysis and Machine Vision*, PWS, Boston, USA, second ed., 1998. ISBN 0-534-95393-X.
 - [52] J. SURI, *Leaking prevention in fast level sets using fuzzy models: an application in mr brain*, IEEE EMBS International Conference on Information Technology Applications in Biomedicine, 2000. Proceedings, (2000), pp. 220–225.
 - [53] J. SURI, K. LIU, S. SINGH, S. LAXMINARAYANA, AND L. REDEN, *Shape recovery algorithms using level sets in 2-d/3-d medical imagery: A state-of-the-art review*, 2001.
 - [54] H. TEK AND B. B. KIMIA, *Volumetric segmentation of medical images by three-dimensional bubbles*, Computer Vision and Image Understanding: CVIU, 65 (1997), pp. 246–258.
 - [55] B. C. VEMURI AND Y. GUO, *Hybrid geometric active models for shape recovery in medical images*, tech. rep., May 19 1998.
 - [56] ———, *Snake pedals: Compact and versatile geometric models with physics-based control*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 445–459.
 - [57] A. K. C. WONG AND P. K. SAHOO, *A gray-level threshold selection method based on maximum entropy principle*, IEEE Transactions on Systems, Man, and Cybernetics, 19 (1989), pp. 866–871.
 - [58] G. XU, E. SEGAWA, AND S. TSUJI, *Robust active contours with insensitive parameters*, Pattern Recognition, 27 (1994), pp. 879–884.
 - [59] L. YATZIV, A. BARTESAGHI, AND G. SAPIRO, *$O(n)$ implementation of the fast marching algorithm*, Feb. 2005.
 - [60] H. ZHAO, T. CHAN, B. MERRIMAN, AND S. OSHER, *A variational level set approach to multiphase motion*, J. Comput. Phys., 127 (1996), pp. 179–195.
 - [61] D. G. ZILL AND M. R. CULLEN, *Differential Equations with Boundary-Value Problems*, Brooks Cole, 2000. ISBN 0-534-38002-6.

Index

- Accuracy
 - segmentation, 88
- BC-Spline, 88
- Bilinear interpolation, 53
- Border
 - FixBorder algorithm, 57
- Bubbles, 20
 - example, 20
- catchment basins, 26
- Constraint energy, 30
- Contrast agent, 70
- Curvature, 43
 - limitations, 44
- Delooping, 31
- Derivative
 - first order backward difference, 41
 - first order centered difference, 41
 - first order forward difference, 41
 - second order centered difference, 42
- Diagnostics, 17
- DICOM file format, 71
- Digital image, 13
- Discretization, 41, 56
 - handling the boundary, 57
- Distance Vector Transform, 80
- Divided difference, 53
- ENO schemes, 51
 - algorithm, 52
 - and isocontour extraction, 53
- Entropy thresholding, 24
- Euler method, 48
- Evolution
 - algorithm, 50
 - in the marker method, 35
- Fast Marching Method
 - algorithm, 61
 - example, 63
 - quadrants, 63
 - quadratic equation, 65
 - solving the quadratic equation, 65
 - update algorithm, 64
 - update rule, 63
- Gaussian speed modifier, 73
- Geometric deformable models
 - bubbles, 20
 - with regularizers, 19
 - without regularizers, 19
- Gradient, 40
- Heap, 66
 - Delete procedure, 67
 - DeleteMin procedure, 68
 - heap condition, 66
 - Insert procedure, 67
 - SiftDown procedure, 67
 - SiftUp procedure, 66
- Implicit function, 38
- Interface, 33
- Isocontour
 - example, 39
 - extraction, 53
- Isodata algorithm, 24
- Laplacian, 43
 - noise, 44
- Level of detail, 14
- Level set
 - discretization, 56
 - example, 39
- Liver
 - segmenting the, 71
- Local thresholding, 25
- Magnetic Resonance Imaging, 70

- comparison with x-ray, 70
 - contrast agents, 70
- Magnetic resonance tomography, 70
- Manual segmentation
 - comparison with automatic segmentation, 89
- Marker method, 34
 - disadvantages, 35
 - example, 34
 - stability, 35
- Model
 - segmentation driven by, 80
- Narrowband Level Set Method, 54
 - example, 55
 - handling the sets, 56
 - problems, 54
 - reinitialization, 55
 - width of the narrow band, 56
- Nearest point on the interface, 40
- Normal vector, 40
 - in the marker method, 34
- Object of interest, 14
- Osculating circle, 42
- OTSU algorithm, 23
- Perfusion area, 80
- Pixel, 14
- Polynomial approximation, 53
- Prototype, 83
 - initialization, 84
 - output, 85
- Region based segmentation, 25
- Region growing, 25
- Region splitting, 26
 - on a quad-tree, 26
- Regularization, 19
- Reinitialization, 57
 - algorithm, 46
- Seed, 71
 - symmetry, 75
- Seed regions, 25
- Segmentation, 13
 - accuracy, 15
 - algorithm overview, 18
 - algorithm taxonomy, 18
 - applications, 17
 - definition, 13
 - FMM example, 74
 - leaking problem, 79
 - problems of FMM in, 74
 - using a model, 80
 - using FMM, 72
- Signed distance function
 - initialization algorithm, 45
 - initialization from an explicit interface, 44
 - reinitialization, 45
- Snakes, 29
 - dual active contour, 31
 - topology adaptive, 31
- Speed function
 - in FMM, 72
 - in levelset segmentation, 75
 - in the marker method, 35
- Surgery planning, 17
- Swallowtail, 36
- T-Snakes, 31
- Threshold, 21
- Threshold speed modifier, 72
- Thresholding
 - adaptive, 23
 - basic, 21
- Topological change
 - in the marker method, 36
- Upwinding, 48
- Visualization, 17
- Volume-of-fluid method
 - advantages, 36
 - disadvantages, 37
 - example, 37
- Voxel, 14
- watershed lines, 28
- Watershed segmentation, 26
 - algorithm, 27
 - example, 28