LEVEL-SET METHODS AND GEODESIC DISTANCE FUNCTIONS

OLA NILSSON

Linköping University
INSTITUTE OF TECHNOLOGY

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden
www.itn.liu.se

Linköping Studies in Science and Technology
Dissertation No. 1275

Cover illustration: Level-sets and distance functions. A color coding illustrating a parameterization for the embedding (sorrounding space) of the the ampersand. The first parameter coordinate is given by the level-sets of the ampersand shape. The second cordinate of the parameterization is given by a distance function along the ampersand curve.

Level-set methods & geodesic distance functions

ISBN 978-91-7393-524-1 ISSN 0345-7524

Printed in Sweden by UniTryck, Linköping, 2009

"This is no longer than fifteen seconds from where we're standing all the way to the bottom.

In between those fifteen seconds you just got to be on it, feel it, and let her buck"

Tanner Hall — *Believe*

ABSTRACT

THIS thesis focuses on *efficient* implementations of *level-set methods* and *geodesic distance functions* in the context of computer graphics (CG). The level-set method (LSM) is a grid based design that inherits many favorable traits from implicit geometry. It is connected to distance functions through its special way of representing geometry: in 3D each point in space stores the closest distance to the surface. A signed distance is used to differentiate between the inside and outside of a closed object. In the discrete form the representation can be thought of as a box around the surface that keeps regularly positioned samples of the distance function. The samples on this *grid* implicitly encode the surface as the zeroth level-set of the signed distance function; hence the name level-set methods. With this representation of geometry follows a toolbox of operations based on partial differential equations (PDE). The solution to these PDEs allows for arbitrary motion and deformation of the surface.

It might seem wasteful to classify the full embedding space (*i.e.* grid domain) just to encode a surface. However, this also gives the possibility to query regions far away from the surface. For applications that deal with space partitioning this is a huge benefit. The existence of a regularly sampled function also makes much of the discrete mathematics relatively uncomplicated.

Today LSM has become widespread in both academia and industry. For some applications, such as the capturing of the air/water interface in free surface fluid simulations LSM is the only realistic choice. In academia highly accurate methods and numerical schemes have been proposed that allow many other areas, like medical imaging, to benefit from LSM and its strengths when it comes to handling topological changes of dynamic curves and surfaces.

Is this level-set method then a silver bullet? As it stands today, not yet. One of its greatest benefits also incurs the major drawback. As a grid based method that samples the distance function over the embedding, rather than over the surface itself, the method is *not space efficient* and because of this *not fast*. This has, to some extent, been amended for by previous work that restricted level-set computations to a thin band around the surface. However, the sheer size of large grids still make LSM impractical for detailed simulations and has rendered the reputation that LSM is memory hungry and slow.

In this thesis we mainly work on efficiency aspects of LSM. We address both grid storage concerns and speed by introducing novel data structures and algorithms that allow for simulations at unprecedented grid resolutions. We also describe a fast method for computing the distance function, which is central to LSM. The thesis also aims at popularizing LSM by showing applications that utilize the new data structures and

algorithms to perform common CG tasks such as parameterization, texturing and rendering.

We first introduce an efficient data structure for representing and dynamically deforming LSM models. The design is called the hierarchical run length encoded grid (H-RLE). It utilizes a space efficient compression of the topology of the geometry model reminiscent of techniques used for storing sparse matrices. The H-RLE features, apart from being compact, fast access to individual grid points: optimal (when amortized) sequential access, and logarithmic random access. Around this versatile data structure applications such as efficient morphing and rendering are built.

Compression and memory efficiency for grid based implicit geometry is further explored in a sequel paper where *both* geometry and topology is compressed in an efficient manner. This paper also thoroughly investigates out-of-core techniques for grid storage and introduces near optimal page replacement and prefetching strategies. Benchmarks show that our work delivers sustainable performance at over 50% of the peak-throughput of a state-of-the-art data structure. At virtually unlimited grid sizes.

To further illustrate the versatility of the level-set method a *reconstruction* method is presented that finds a physically plausible surface from sparse contour input data. We demonstrate the robustness of the method on a variety of medical, topographic and synthetic data sets.

In LSM the surface is defined by a distance function. Much work has been dedicated to finding such distance functions on different classes of geometry representations using discretizations based on PDEs. In general there is no differentiable solution to this type of problem and unique weak solutions are selected using the vanishing viscosity condition. In this work we adopt and generalize a method that computes distances over curved manifolds (surfaces and volumes) without relying on differentiability. The discretization is based on the dynamic programming principle and an integral definition of geodesic distance. The input assumes a sampled metric tensor field and a tessellation of triangles or tetrahedra. Our method is simple to implement and has an intuitive geometric construction. Benchmarks show our method to be more exact and have better convergence properties than competing work without sacrificing speed. In addition the method works for arbitrary Riemannian manifolds while retaining both simplicity and efficiency.

An obvious drawback of implicit surfaces when used for CG is the lack of a natural parameterization. In this setting it is natural to leverage the properties of geodesic paths on surfaces because of their minimizing qualities. Previously published work in this area find parameterizations of the surface using geodesic distance in an ad-hoc manner. Either by dividing the surface into patches that are iteratively optimized, or by using distances to approximate the (in differential geometry) well known exponential map. We present a novel method for parameterizing surfaces that computes *the logarithmic map* (the inverse of the exponential map) over triangle meshes. The method, however, easily translates to any setting in which distances can be computed. Our results show less distortion and the proposed method is faster than competing work.

PREFACE

OVER the last five years I have had the opportunity to work intensely and wholeheartedly with computer graphics. I started my Ph.D. at Linköping University (LiU) in the spring of 2004 and have since then been privileged to research, study, present, learn, be inspired, and teach, in various positions, in various locations, around the world.

When I first started I was confronted with an american professor of danish origin who said: "Here's the ball. Run with it!" This turned out to be a very accurate description of what was to come. My supervisor – professor Ken Museth – was, at the time, head of the graphics group at Linköping University and also held an adjunct position at Aarhus University where part of the group was located.

The ball that was pitched was an idea about doing segmentation by contour metamorphosis with level-set methods. It was a challenging project, and I was working hard to make the IEEE Visualization conference deadline for 2005. When the paper [62] was accepted I was happy as a clam. Moreover, our project concerning efficient data structures for level-set methods headed by Ph.D. student Michael B. Nielsen at Aarhus University led to a publication [40] at the prestigious ACM SIGGRAPH conference the same fall. The work was a joint project together with Canadian visual effects company Frantic Films and was presented as a sketch, or abstract presentation, by Michael B. Nielsen and Ben Houston. The full paper [41] was later published in ACM Transactions on graphics (TOG).

Later that year, during the summer, my supervisor Ken Museth and I, with appreciated support from Reiner Lenz at LiU, got the opportunity to travel to Tokyo, Japan, and visit the Miyaki Laboratory at Chiba University. The trip was both interesting and inspiring. The very first thing that met us in Japan was an experimental study on computer graphics in medical environments. In this case several surgical operations were conducted on a sedated pig in one of the class rooms at the campus. The experience was surreal.

During the stay in Japan I started the research leading to the proposed method for computing simulations on manifolds (see [61]) and especially reaction-diffusion equations.

The year of 2005 was concluded with a conference presentation in Minneapolis, USA, in front of an interested audience at VIS 2005. A busy year indeed, and a great start for my Ph.D. degree.

In the spring of 2006 I got a position as visiting researcher at CMA at the University of Oslo. During this time I took an excellent course on splines theory and simultaneously worked on the compression/out-of-core project that led to publication [61].

The project included a rather involved coordination of logistics. At the time project members were stationed in Oslo, Aarhus, and Los Angeles.

During the spring of 2007 my fellow Ph.D. students Gunnar Johansson, Andreas Söderström and I took over the course "Modeling and animation" after Ken Museth who was on a leave of absence. This involved a large amount of work. We had a good foundation from previous years lecture notes but decided to completely rework the practical parts. This worked out very well, and in the end the course was voted outstanding by the students awarding us with an acknowledgment from the Dean.

At the same time, I started to collaborate with Anders Brun, then at Linköping University. He had made some progress in parameterizations using distance functions but was in need of more accurate distance estimates. I had a large range of implemented distance computation methods available and we teamed up. Parameterization is a challenging field and the needs in computer graphics additionally throw in special constraints; the project was put on ice for the most of 2007. Eventually, this project led to publication [16], with Anders and I as mutual first authors.

For an exciting and intense period during the fall of 2007 I was interning at visual effects company Digital Domain (DD) in Venice, Los Angeles. Initially, I was working on fast methods for conditioning the level-set scalar fields, but the main project for my stay in Venice focused on aspects on vector field data in fluid simulations. Due to the limited amount of time the project was not finished, however, software I developed at DD has since been used in production.

Over the year of 2008, my situation changed dramatically both professionally and personally. I became a father to a wonderful daughter – the joy of my life. But sadly, Ken Museth decided to leave Linköping University. To my relief, Ken agreed to stay on as adjunct professor and committed to keep supervising me. Another surprising event was that our paper [61] was especially invited for presentation at the ACM SIGGRAPH conference.

The parameterization project with Anders Brun involved a lot of work on distance computations. We quickly realized that commonly used methods, despite their reputation, were not good enough. Together with Martin Reimers at Oslo University we developed a method for computing accurate distance functions in flat and curved spaces. This led to publications [63] and [64], see also Chapter 6.

ACKNOWLEDGMENTS

THIS thesis would not have been written had it not been for a number of fortuitous events. That my supervisor professor Ken Museth would find Linköping University a worthy sequel to Caltech, is at the very core. He has shown me the world of computer graphics, and especially level-set methods, with an unrivaled enthusiasm and dedication. Thank you.

~

Many people have helped me during this time. Among them my family comes first. Thank you Ingebjørg for reading my text carefully, and helping me recognize what's really important.

~

I would also like to acknowledge:

The research group that Ken created – the graphics group at LiU– an exciting and extremely useful resource. To meet and collaborate with the Aarhus connection has been a pleasure. Michael Bang Nielsen and Anders Brodersen became not only colleagues but also good friends. Part of this work is based on their research. The graphics group Ph.D. students in Norrköping: Gunnar Johansson and Andreas Söderström. Thank you all for proofreading and support.

The staff at the department (ITN/DM), especially Reiner Lenz, Björn Gudmundsson, and Gun-Britt Löfgren.

Professor David Breen at Drexel University.

Professor Tom Lyche, Helge Galdal, and Martin Reimers made my stay at Oslo University possible. The CMA group and especially the Ph.D. students Pål Hermunn Johanssen, Torquil MacDonald Sørensen, and Tore G. Halvorsen made it fun. Martin Reimers deserves an extra sincere thanks for the work he put in into collaboration/supervision.

Anders Brun has over the last two years in many ways acted as a secondary supervisor. Together we have explored many aspects of distance, geometry and parameterization. It's been fun, challenging, and rewarding.

The crew at Digital Domain: Craig Zerouni and Doug Roble who made my internship possible. Doug Roble also made an appreciated stand-in as secondary supervisor and guided me through the magic of fluid simulation. Mattias Bergbom, Charles-Felix Chabert, and Mårten Larsson were all part of the wonderful experience of Venice, both professionally and socially.

APPLICATIONS GALLERY

I HAVE produced a significant amount of images using the methods and software that I have developed for this thesis. In this teaser gallery a small sub-set is shown to illustrate some of the more visual applications of this work. All the models are represented as level-set data except when specified otherwise.

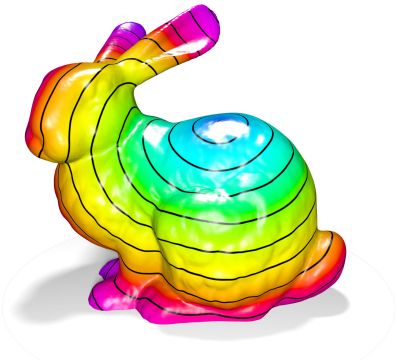


Figure 1: Visualization of a distance map on the "Stanford Bunny". See article [64] and Chapter 6.

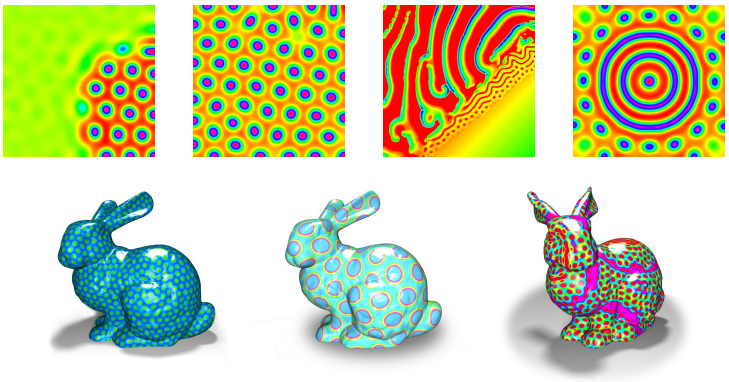


Figure 2: Growing 2D textures on curved surfaces. Top row: Turing's reaction diffusion equations in the plane. Bottom row: textures grown directly on the curved surfaces using the technique presented in [61].

Figure 3: Direct rendering of implicit geometry represented by the level-set method as described in [41]. Winged horse model constructed with smooth CSG operations, courtesy of Henrik Wrangel.

Figure 4: The wave equation ($\partial^2 f/\partial t^2 = c^2 \nabla^2 f$) describes the propagation of waves in the plane. Here solved over a curved surface, also using the technique in [61]. The color coding shows the propagation of the waves at three different time steps.

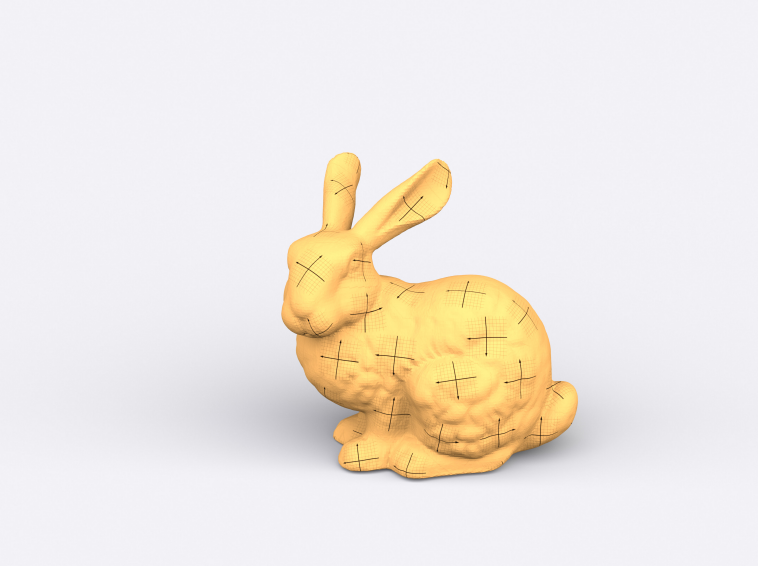


Figure 5: Local texture mapping on the Stanford Bunny (triangle mesh) using the technique from article [16].

CONTENTS

CHAPTER 1

INTRODUCTION

In computer graphics (CG), triangle meshes have been the prevalent choice of surface representation for a long time. Triangles have several benefits: rendering is very well supported in hardware, typical graphics math is well studied and implementable, and storage and computational complexity is relatively small. In this thesis we promote an alternative geometry representation for use in CG, namely implicit geometry. More specifically a special variant thereof: the level-set method (LSM).

Triangle meshes can be seen as an approximation of an underlying smooth surface. But, then one may ask: What is a surface? And how does one formalize the subject? It appears that this is a well studied topic in mathematics. In the discipline of differential geometry a surface is called a *manifold*, and is characterized by certain properties. Usually some sort of smoothness, or existence of derivatives, is required. A matter in question is that these derivatives are cumbersome to define on an irregular triangle mesh[1]. It is even not generally agreed on how to do this in some cases. Another problem is that splitting and merging triangle meshes is difficult, especially when the surface moves.

On the other hand, an implicit geometry specification does not suffer from these problems. Derivatives are readily available, and topological changes are resolved almost automatically. Implicit geometry works by categorizing all space into *inside* and *outside* making partitioning problems straightforward. It is, in fact, impossible to arrive at ambiguities when using implicit geometry. These benefits makes an implicit setting attractive for many applications.

Implicit geometry is well known through the study of implicit functions in mathematics. However, in computer graphics implicit geometry has not been very popular, except for special cases, and the applications have been more proof of concept than practical tools. With the invention of the level-set method (LSM) in 1988 [66] this started to change, and recently LSM has gained a large amount of attention in the computer graphics community [31, 53, 57]. The level-set method is a discretized version of implicit geometry that additionally provides a full toolbox for dealing with moving surfaces, or generally, manifolds. Through its special representation, LSM also gives a way of trivially measuring the closest distance between objects in space.

Unfortunately, there is no such thing as a free lunch and the level-set method also has downsides. In this thesis we address some of the concerns with the method. Our work exploits the positive traits of implicit geometry and LSM in an efficient manner

1. This is an active and promising area of research but still not as mature as implicit geometry.

and we display advanced and original results. We also try to popularize the method by showing how to do standard computer graphics operations in the level-set formalism. We do stress, however, that the level-set method is not a "silver bullet". In some cases LSM is the only realistic choice but in many situations the decision of representation needs careful consideration.

We start the thesis with some inspiring examples from the visual effects industry showing LSM in use. Then, we isolate some disadvantages with the method and indicate our solutions – justifying this work. This is followed by a list of the included publications and their respective contributions. We also point out some of the potential impacts of this work. The introduction chapter is finally concluded with an overview of the thesis.

1.1 MOTIVATION

The motion of water is a most fascinating scenery. The rich behavior of the fluid flow with waves, wakes, and eddies as well as the complex merging and pinching of droplets and surface can be truly mesmerizing. Today the artificial simulation and rendering of water has reached incredible realism; it is very difficult to discern simulation and visual effects from live-action in contemporary motion pictures. Figure 1.1 show shots from several motion pictures illustrating the progressive trend. The simulation of fluid flow – or computational fluid dynamics (CFD) – is a well studied discipline that dates back a long time in the fields of physics and applied mathematics. Highly accurate numerical schemes that handle turbulent flows as well as the presence of shocks and discontinuities exist, *cf.* [4].

For computer graphics, the efforts in CFD have been focused more on realism and efficiency than on accuracy. A typical example is fluid simulations for visual effects in motion pictures. In this scenario, the director is more interested in the behavior of the fluid surface than the accuracy of the actual fluid flow, as long as the flow is realistic enough. And the producer is even more interested in how fast the simulation runs to keep costs down. When simulation is used as a modeling tool fast turnaround times are also necessary for a more creative reason; artists generally do not like to wait.

A major hurdle in the simulation of free surface flows is the tracking of the fluid surface. For this complex and difficult task LSM is an ideal candidate and its introduction has proved to be a key advancement in the field. One of the first major motion pictures featuring fluid flows captured with the level-set method was *Lord of The Rings: The Fellowship of the Ring*. The scene where the "ringwraiths" are trying to cross the river and the water rises is shown in figure 1.1(a). A couple of years later, in the motion picture *The Day after Tomorrow*, fluid simulation in visual effects had something of a breakthrough. Figure 1.1(b) shows one of the key scenes where a giant wave crashes on New York. These two early examples contain, in addition to raw simulation data, heavy post-processing. The horses in (a) are not simulated for example, nor is the foam in neither of (a) nor (b).

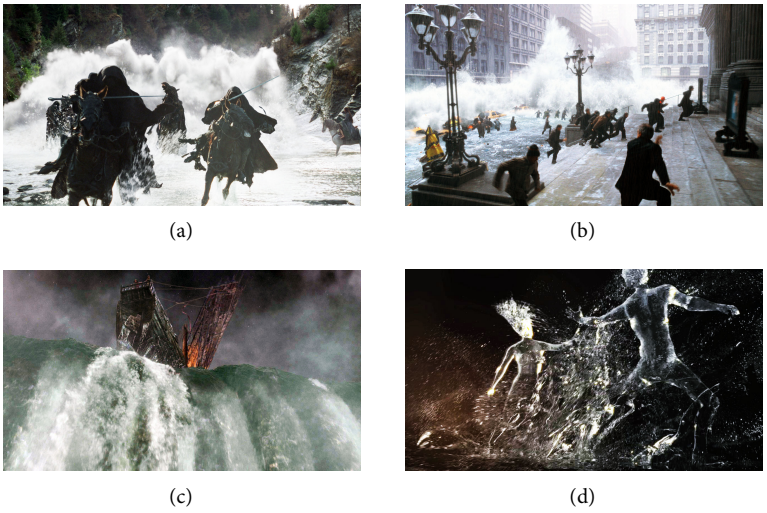


Figure 1.1: Some examples of fluid simulations in visual effects for motion pictures and commercials. All water surfaces are captured with the level-set method. (a) *Lord of the Rings: The Fellowship of the Ring*, 2001. ©New Line Cinema (b) *The Day after Tomorrow*, 2004. ©Twentieth Century Fox (c) *Pirates of the Caribbean: At World's End*, 2007. ©Walt Disney Pictures (d) *Sundance* – a Bacardi commercial, 2008. Images courtesy of Digital Domain.

The quality of the simulations for these early movies was restricted due to space and efficiency constraints. In more recent movies larger simulation domains, *i.e.* larger grids, are deployed leading to more detailed simulations. See for example sub figures (c) and (d), where special data structures [52, 55, 56, 59] make high resolution simulations possible.

1.1.1 Other uses

The level-set method is used in many other contexts apart from the high profile applications in visual effects presented above. In medical imaging, for example, the capturing of interfaces translates directly to a segmentation problem. The act of differentiating between cancer tumor and healthy tissue can be formulated as a space partitioning problem, which is ideal to solve with LSM. Another kind of segmentation is shown in Figure 1.2 where the location of blood vessels are extracted in an image of a retina.

Other areas using LSM include, but are not limited to, geometric modeling [13, 54], shape morphing [8], and volume segmentation [18]. The books [65, 75] provide ample examples and areas of use for the method.

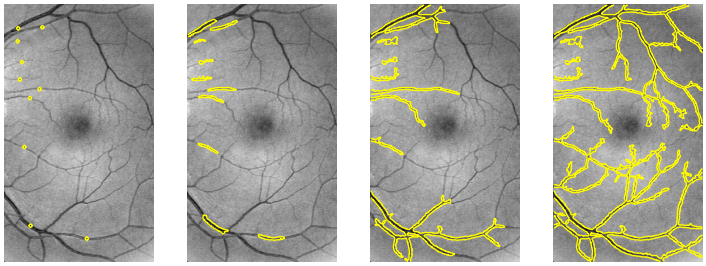


Figure 1.2: Implicit geometry in a medical imaging, LSM segmentation of blood vessels in the retina by iterative evolution of an initial guess. The number of performed iterations is listed below each figure. Images courtesy of Gunnar Läthén.

1.2 KEY ISSUES

We have already indicated that there exists some disadvantages with the level-set method. The work in this thesis was initiated as a response to these limitations and provides solutions to some of the problems with LSM, as it stands today.

The fidelity of a simulation is, if well constructed, restricted by the size of the simulation domain, *i.e.* the size of the grid, and the quality of the numerical schemes used. This relation provides a direct connection between grid sizes and simulation quality. Traditionally, the representation of a surface with LSM has been inefficient in terms of both storage and efficiency and this bottleneck has restricted simulation quality. Thus the first identified issue is:

1. the need to represent larger simulation domains, *i.e.* high resolution grids.

This should not come at the prize of a slower representation, which leads us to the following sub-problem

   (a) the representation must carefully weigh storage requirements against efficiency.

Ideally, the representation should be configurable with respect to these requirements. Our solution introduces loss-less compression of the topology of a model. In addition, lossy compression of the geometry, coupled with out-of-core techniques provide efficient ways of representing LSM geometry in a versatile and configurable way.

In the context of computer graphics one of the major downsides to the level-set method is that

2. being an implicit geometry specification LSM lacks a natural parameterization.

This might not be evident when looking at the examples shown so far. Water, for example, is readily shaded using a glass material. However, there is no simple way of

performing texture mapping on the (implicit) geometry. This makes the representation inadequate in many CG applications unless first converted into a form that can be easily parameterized. Because of this one of the main efforts of this work has been develop a robust and efficient parameterization method applicable to implicit geometry.

The level-set method, and our proposed way of parameterization, are tightly coupled to the notion of geodesic (closest) distance. Several schemes for computing distance in the context of LSM exist. But, none of these methods provide sufficient accuracy (coupled with speed) to suit our needs. We show that

3. a linearization of the distance function is not accurate enough for some applications.

Instead, we adopt a method originally designed for triangle meshes into a grid based setting. This method is then extended to handle anisotropic domains, 3D, and different types of boundary conditions.

The interest in level-set methods, distances and parameterizations was introduced through the opening project of the PhD. Motivated by actual medicine work-flow we presented a robust and simple reconstruction method based on distances and contours with the aim to reconstruct a three-dimensional volume from a set of sparsely sampled contour curves using the level-set method.

The identified issues above and their proposed solutions form the basis for the work in this thesis and correspond more or less directly to the different included articles.

1.3 PUBLICATIONS

The publications that are included in this thesis are

Article 1 – [62] O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis, an Eulerian approach with Lagrangian particle tracking. *Visualization, 2005. VIS 05. IEEE*, pages 407–414, 2005

Article 2 – [41] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.*, 25(1):151–175, 2006

Article 3 – [61] M. B. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set methods. *ACM Trans. Graph.*, 26(4):16, 2007

Article 4 – [16] A. Brun, O. Nilsson, M. Reimers, K. Museth, and H. Knutsson. Computing Riemannian normal coordinates on triangle meshes. *In submission*, 2009

Article 5 – [64] O. Nilsson, M. Reimers, K. Museth, and A. Brun. Efficient computations of geodesic distance. *In submission*, 2009

The following publications are related to this work but not included: [40, 45, 60, 63].

1.4 CONTRIBUTIONS

I have done a substantial amount of work for all of the publications included in this thesis. I am the primary author of the work in Article 1, and 5 [62, 64], and for these articles I have developed the majority of both ideas and code. This is also true for the preliminary results presented In Chapter 6 and 7.

In article 4 [16] Anders Brun and I have mutually shared the workload and are both considered first authors.

When it comes to the LSM data structure projects, that is paper 2 and 3 [41, 61], I worked as a team member developing both code and ideas.

A brief summary of the contributions in the included papers is given below.

Article 1 – [62] introduces a generic and intuitive method for reconstructing 3D volumes from 2D input contours. The procedure uses metamorphosis between consecutive contours to find a physically plausible 3D shape.

Article 2 and 3 – [41, 61] present efficient ways of representing geometry using LSM. The representations are configurable in that several levels of compression can be applied depending on the requirements on speed. At the highest level both topology and geometry is compressed resulting in an extremely light memory footprint. On the other end of the range only topology is compressed while fast memory access is retained.

Article 4 – [16] shows how to perform a mathematically sound and efficient approximation of the logarithmic map over triangle meshes. However, the method translates to any setting in which distances can be measured. The resulting maps are proposed for usage in local texture mapping and decal compositing.

Article 5 – [64] gives an way of computing distances over anisotropic domains. The method is efficient and straightforward to implement. In addition, it has an intuitive geometric construction.

Chapter 6 Extends the work on article 5 ([64]) and shows accurate distance computations in anisotropic 3D domains.

Chapter 7 Proposes a faster way of computing the parameterization discussed in article 4 ([16]).

1.5 POTENTIAL IMPACT

1.5.1 High resolution LSM simulations

The ever increasing resolution in measuring equipment in medical science (*e.g.* better MRI- and CT-scanners) gives larger and larger data sets to work with. As do the escalation of digital sensor quality in film and photography. The novel television

broadcast standards additionally inflate the number of pixels. Consequently, increasing resolution for virtual imagery is needed as well.

The introduction of data structures and algorithms developed in this thesis has effectively lifted some of the previous constraints on grid sizes when using LSM. Taking advantage of larger grids means access to more detailed simulation domains. This naturally opens up for better results whether the application is medical segmentation or fluid simulation.

The impact of the new data structures is already notable. Software developed for this thesis has already been used in major motion pictures and the level-set data structures are in use in visual effects production. Our work is also acknowledged by academia and used in different contexts where high resolution level-set simulations are of importance.

1.5.2 Vector space algorithms

Our proposed parameterization method produces *Riemannian normal coordinates* (RNC) – a canonical coordinate system with many applications in differential geometry. One of its uses in computer graphics is to translate algorithms from vector spaces to surfaces. Thus, for instance, accurate estimation of RNC over a surface mesh enables users to implement classical vector space algorithms on triangulated surfaces. This includes important methods such as K-means clustering and mean shift clustering. The future use of our method as a building block for algorithms and interactive tools is also helped by the fact that it is straightforward to implement.

The RNC parameterization is, as mentioned, tightly coupled to the notion of geodesic distance. Our proposed distance solver for Riemannian manifolds also enables many known algorithms to be generalized to a *non-linear* geodesic setting with little effort. This includes classical tools such as skeletonization, dilation/erosion and watershed segmentation.

1.5.3 Popularizing LSM

The level-set method provides an efficient and natural way to handle dynamic surfaces and general interface problems. Its status in CG has been somewhat subdued by a reputation of the method as being slow, impractical for CG usage, and cumbersome to implement. This thesis shows several examples where LSM is used in computer graphics instead of the traditional triangle mesh. This includes rendering, PDE's on surfaces, texturing, and more. The impact of this work, and others like it, is a positive change in attitude regarding implicit surfaces and LSM in computer graphics. The effect is noticeable in the last years' increasing interest in the level-set method.

1.6 DISSERTATION OVERVIEW

The thesis can be divided into several parts. The first three chapters are introductory but also define the notation used throughout the text. The intended reader is not

acquainted with differential geometry, surfaces or LSM. Knowledgeable readers can safely skip these chapters.

Chapter 2 introduces some of the concepts needed when discussing surfaces and geometry from a mathematical point of view.

Chapter 3 presents the two main geometry representations used in this thesis: implicit and explicit geometry.

Chapter 4 continues to describe the theory of level-set methods as a special branch of implicit geometry.

After this, the focus of the text changes towards computer science and hardware related issues. Readers curious of the different types of data structures presented in the thesis should be interested in the following chapter.

Chapter 5 describes implementation specific issues for LSM and tries to summarize how to arrive at the different sparse data structures described in papers [41, 61].

Then, the next two chapters report preliminary results in the area of distance computations and parameterization. The reader is encouraged, but not obliged, to first read the corresponding included article.

Chapter 6 extends some of the ideas explored in article [64]. In particular, a method for computing geodesic distance in 3D is presented.

Chapter 7 is a continuation of the ideas from paper [16]. The chapter presents an alternative, faster, way of computing the parameterization.

Finally, an appendix contains the articles included in the dissertation.

CHAPTER 2

SURFACES AND MANIFOLDS

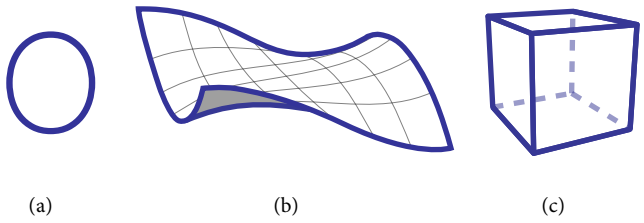


Figure 2.1: A curve, a surface and a volume.

Above, three different types of geometry are shown. To the left is a curve, in the middle is a surface, and to the right a volume is shown. This thesis shows how the level-set method can represent and manipulate these and other kinds of geometry in a versatile and efficient manner. But before we start looking into how to do this, we need to specify what exactly do we mean by geometry? What properties are important, and how is the subject formalized? This introductory chapter is dedicated to answering some of these questions and prepare the reader for the upcoming discussions.

2.1 INTRODUCTION

To begin with let us distinguish between *geometric shape* and *connectivity*. Shape is a property that describes the appearance of an object or more formally how the object is curved (or not curved). Connectivity on the other hand concerns the way an object is put together. Already in the 18:th century Leonhard Euler realized that there are problems that deal not with the distance between objects but rely solely on their connectivity. The famous "Seven bridges of Königsberg" problem asks if it's possible to walk the city of Königsberg and cross each of its seven bridges once and only once. Euler separated the riddle into two disjoint problem classes: 1) the exact positions of the bridges, and 2) how the bridges connect the mainland with the islands. He then realized that the solution to the original question is only depending on the second class, that is the connectivity.

The riddle would today be categorized as an example from the field of *topology* and Euler's particular way of solving it would use graph theory. Topology is a vast mathematical discipline that deals with connectivity of space on both large and small scales. Sometimes this is referred to as the fine and global structure of space. A thorough introduction to topology is beyond the scope of this thesis, we merely present important terms and concepts on a need to know basis.

Geometric shape falls in the category of differential calculus, or more generally *differential geometry* which is generalized to curved spaces. In this chapter we introduce some of the basic usage for curves, surfaces and volumes. The derivative is a key concept, and will be used to describe the shape and smoothness of an object. Differential geometry and topology are closely related and sometimes difficult to tell apart. For that reason we will try to use the simpler and more descriptive words geometric shape and connectivity where appropriate.

2.2 TOPOLOGY

A curve is a one dimensional geometric object in two dimensions or more. In Figure 2.1(a) the closed plane curve is actually the lowercase letter 'o' in the typeface Helvetica Neue Ultralight. An important categorization from topology states that this 'o'-curve is equal to a '0'-curve (the number zero). It is in fact equal to any closed non-intersecting curve in the sense that they all separate the plane into an inside and an outside. This we realize intuitively, and it is formally stated in the *Jordan curve separation theorem*[3]: any simple closed curve separates space into exactly three regions: an inside (which is finite if the curve is bounded), an outside, and the curve itself. These letter curves share the same connectivity and are *homeomorphic*. The letter 'Q' however is not equal, since it has a tail. In the same reasoning the surface in Figure 2.1(b) is equal to a flat plane and the cube in (c) equal to a sphere.

A surface is an example of two dimensional geometry. Normally, a surface is thought of as living in a three dimensional space. This is most natural for us as it resembles "reality" as we perceive it. This is also called an *extrinsic* view point, since the geometry is "observed" in a higher dimension. Consider instead an ant living on the surface of Figure 2.1(b). The ant has (supposedly) no idea of the third dimension – it thinks it lives in a two dimensional world. The ant has what is called the *intrinsic* view. Both viewpoints have benefits and drawbacks.

If a surface is closed it encloses a volume, like the volume in Figure 2.1(c) is bounded by its six faces.

2.2.1 *Categorizing topology*

Since we will be working with dynamic geometry we also need to address what happens when pieces of geometry merge as shown in Figure 2.2, or conversely break apart. The

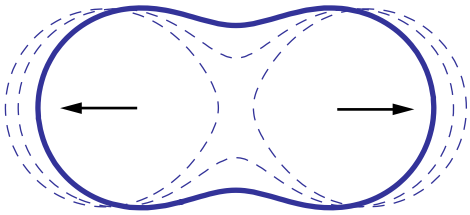


Figure 2.2: A single object breaking up into two disjoint ones.

Euler-Poincaré characteristic χ is an invariant used to describe the connectivity of an object irrespective of how it is deformed (homeomorphically).

$$\chi = V - E + F \tag{2.1}$$

Here V is the number of vertices, E is the number of edges and F is the number of faces. To determine the characteristic of a general object it is often most straightforward to find a polygonization of the object and then apply the above formula. Otherwise the *Gauss-Bonnet* theorem [24] connects the geometric shape of a two-dimensional manifold with its topology. This is done by coupling the curvature of the surface to the Euler-Poincaré characteristic through the following integral over the manifold

$$\int_M K_g \, dA = 2\pi\chi. \tag{2.2}$$

Here, K_g is the Gaussian curvature as explained in 2.4.2. To exemplify, Equation (2.2) finds the Euler-Poincaré characteristic of a sphere to be 2, since the sphere has constant Gaussian curvature $K_g = 1/r^2$. This is the same as the characteristic for an ellipse, a box, or any homeomorphic closed surface.

Another invariant is the *genus* of an object, which translates to the number of "handles", or more formally how many closed curves can be cut through the object without disconnecting any parts. It is related to the Euler characteristic by the following equation

$$\chi = 2 - 2g - b, \tag{2.3}$$

where b is the number of boundary components and g is the genus. The sphere (and any closed non-intersecting surface) consequently has genus zero. This agrees with having no handles.

2.3 MANIFOLDS

We have so far described geometry in terms of graphical concepts. To formalize the subject we introduce a fundamental term from mathematical topology – the *manifold* – which is used to characterize space. A somewhat loose definition says that:

A manifold M is a mathematical space that is locally homeomorphic to Euclidean space.

This means that the two constructs are topologically equivalent in the sense that in a small neighborhood the manifold resembles Euclidean (flat) space. A manifold can be thought of as a curved version of conventional space. That is the line generalizes to a curve and the plane to a (curved) surface. The dimensionality of the manifold is equal to the corresponding Euclidean space; a line or curve is a one dimensional manifold, a plane or surface is two dimensional, and so forth.

A familiar example of a two dimensional manifold is the sphere, which is a curved closed surface in 3D. Locally a neighborhood of the sphere can be mapped to a plane, as is done in an atlas of the world. On a global scale, on the other hand, the sphere and plane are not topologically equivalent.

2.3.1 *Classification*

An important consequence of the definition is that a manifold cannot contain holes, it must be "watertight", apart from at a boundary. A manifold can neither contain "junctions". Both these concepts are illustrated in Figure 2.3. Any closed non intersecting curve (a) is a one dimensional manifold since it resembles a straight line locally. The homeomorphism can be generically verified with the aid of a sufficiently small n-sphere. From the definition above we know that the manifold must resemble Euclidean space locally. We formalize this criteria by considering the neighborhood $\boldsymbol{N}$ of any point in the manifold and require this to resemble an open Euclidean n-sphere,

$$\boldsymbol{N}^n = \{\boldsymbol{x} \in \mathbb{R}^n \mid \textstyle\sum x_i^2 < 1\}. \tag{2.4}$$

For a one-dimensional manifold this simply means local likeness to a line. We can use a sufficiently small circle for the check; it must be possible to place the circle anywhere over the curve with *exactly* two intersections.

The boundary, or edge, of an n-dimensional manifold is an $(n-1)$-dimensional manifold, we denote it with ∂M. Therefore the boundary must be homeomorphic to a "half" n-sphere, or Equation (2.4) with the additional constraint: $x_1 \geq 0$. Thus an open curve is a one dimensional manifold with a boundary (b). Using the circle test we now also require a strict single intersection at the boundary. Counter example to the 1D manifolds is (d) – the closed zero-set of the cone function. At the origin it is impossible to place even an infinitesimal circle without more than two intersections.

The torus, or donut, (c) is a two dimensional manifold because at any place a disk placed on the surface crosses no edge. Similar to previous reasoning makes the level-set (iso-value = 0.1) of the clipped cone function in (e) a two dimensional manifold with a border; a disk crosses no edge except at the boundary where it crosses exactly one edge (two intersections). A 2D counter example is the ball-plane compound in (f) where a disk can be placed such that the edge is intersected only once.

A manifold can furthermore be categorized as being bounded meaning that it has finite size, or unbounded. The circle is an example of a closed bounded geometry

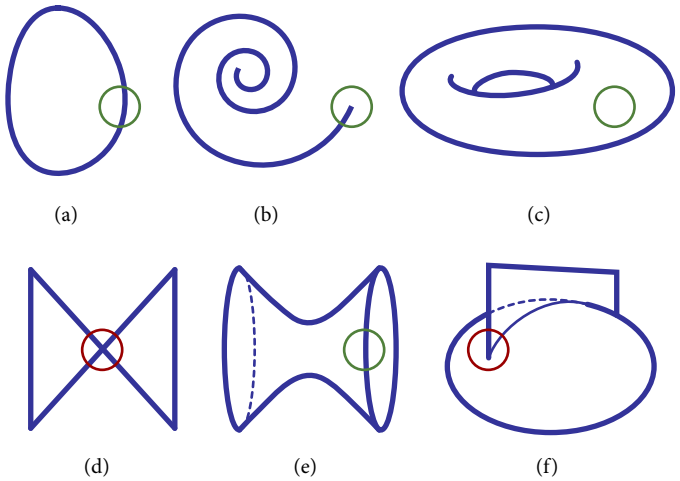


Figure 2.3: Manifold examples and counter examples. (a) 1D manifold (b) 1D manifold with boundary. (c) a 2D manifold. (d) a 1D non-manifold. (e) a 2D manifold with boundary. (f) a 2D non-manifold.

whereas a plane is unbounded. A manifold that is closed and bounded is said to be *compact*.

2.3.2 *Manifolds and atlases*

Following the analogy of an atlas of the earth we introduce the terms *charts* and *atlases* as used in mathematics. Let the manifold M be defined by a set of open subsets $U_i \subset \mathbb{R}^n$. For each of the subsets an invertible map called a *chart* exists. We denote the chart with Φ and require it to be a homeomorphism onto an open subset of Euclidean space

$$\Phi_i : U_i \longrightarrow \mathbb{R}^n. \tag{2.5}$$

This is illustrated in Figure 2.4. From the definition we know that the manifold is encompassed by the subsets and hence we also require the charts to cover the manifold[1]. Such a collection of charts is called an *atlas*. The charts may overlap. Using the atlas which holds the connectivity of the charts it is possible to transition from chart to chart and thus "navigate" around the manifold.

An atlas can be charged with additional structure such as differentiability which we will address in section 2.4.

1. Normally this calls for more than one chart, but not necessarily.

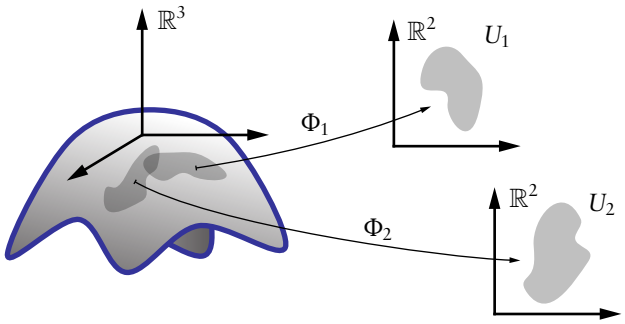


Figure 2.4: Two charts from an atlas of a manifold overlap and allow us to transition between subsets.

Figure 2.5: The Möbius strip (a) is an non orientable surface whereas the torus (b) is orientable or two sided.

2.3.3 *Oriented surfaces*

The special case of two dimensional manifolds can, in addition, be classified as *orientable*. This means that they are two sided, in contrast to non-orientable (one sided) surfaces. Typical examples of non-orientable surfaces are the *Möbius strip* (see Figure 2.5) and the *Klein bottle*. A surface is non-orientable if it is possible to trace an arbitrary curve from a point $\boldsymbol{p}$ along the surface and back in such a way that the surface normal changes direction. All other surfaces are orientable. From this we can directly deduce that all closed non-intersecting surfaces are orientable.

Non-orientable surfaces are important in topology and theoretical geometry. In this work we focus mainly on geometry that is orientable, which also means that it is *physically realizable*, such as the torus (Figure 2.5).

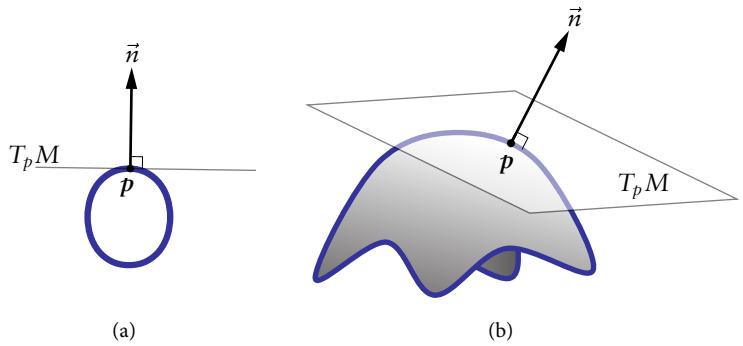


Figure 2.6: The normal vector $\vec{n}$ at point $\boldsymbol{p}$ is perpendicular to the tangent space $T_{\boldsymbol{p}}M$. This is illustrated for curves (a) and surfaces (b).

2.4 DIFFERENTIABLE MANIFOLDS

A *differentiable manifold* is a manifold for which the atlas is charged with a differentiable structure. This means that the local neighborhoods are compatible enough to do calculus and, for example, find derivatives in the manifold. Derivatives can then be used to classify the shape of a manifold, and consequently its smoothness.

A differentiable manifold also guarantees the existence of tangent spaces for all points $\boldsymbol{p}$, denoted by $T_{\boldsymbol{p}}M$. The tangent space at $\boldsymbol{p}$ is spanned by the directional derivatives along all curves passing through the point, $\partial/\partial x_i$. Some tangent spaces are shown in Figure 2.6.

2.4.1 *The normal vector*

For two- and three-dimensional manifolds the existence of the tangent space can be used to define the *tangent* and *normal* vectors, as shown in Figure 2.6. The tangent space for a curve is one-dimensional and defines the direction of the tangent and normal vector uniquely (a). For a surface the tangent space is a plane $T_{\boldsymbol{p}}M$ which uniquely defines the normal vector. The tangent vectors are any two arbitrary unit vectors that span $T_{\boldsymbol{p}}M$ (b).

Since the tangent space is spanned by the first order derivatives the normal and tangent vectors define the local shape of an object at each point. As a whole this gives a linear approximation of the objects shape. In order to get more information, we need to evaluate higher order derivatives.

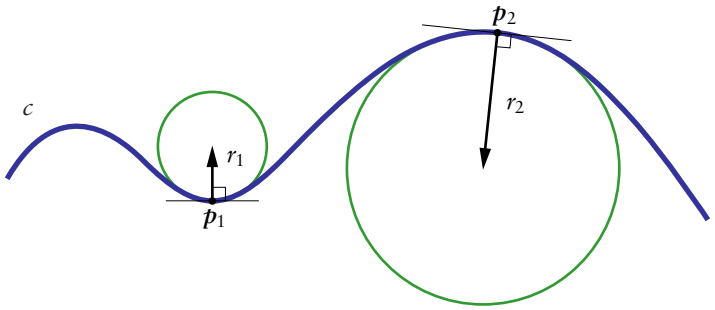


Figure 2.7: A planar curve c with points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and their respective osculating circles. The curvature at $\boldsymbol{p}_i$ is the reciprocal of the radius of the *osculating* circle, $\kappa = 1/r$.

2.4.2 *Curvature*

The degree to which something is curved is called *curvature*. It is straightforward to define for a curve in the plane. It can be interpreted as any of the following equal concepts:

> *inverse of degree of flatness, reciprocal of the radius of the osculating circle, or rate of change of the tangent direction.*

In Figure 2.7 the notion of the osculating circle is depicted. It is the largest possible touching circle with equal tangent vector at the point of contact.

Curvature is furthermore a second order effect, so we must assume the curve to be twice differentiable.

Curvature for surfaces

For surfaces – and in higher dimensions – it gets more complicated. Measuring the degree of flatness or deviation from the tangent plane can be done in several ways. When extending the curvature definition for planar curves into 3D it is evident that the curvature at a point no longer is unique. For any surface point there exists many planar curves all with potentially different curvatures. Without loss of generality we let each of these curves be confined to a plane containing the surface normal at the point. These planar subsets are called *normal planes*, and the curves are the corresponding plane curves. Some sample planar curves for a "monkey saddle" surface are shown in Figure 2.8. Now each of these curves, c_i, has a corresponding curvature, κ_i, as previously defined. From all these curvatures two κ_i can be used to characterize a surface in an efficient way and play an important role in many geometry applications. Let the *principal curvatures* be the maximum and minimum curvatures of the plane curves at $\boldsymbol{p}$ and call them κ_1 and κ_2. Then, the *Gaussian curvature* is the product of the principal curvatures

$$K_G = \kappa_1 \kappa_2. \tag{2.6}$$

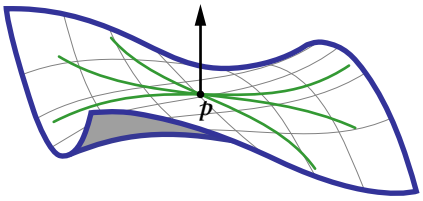


Figure 2.8: A parametric surface, $z = x^3 - 3xy^2$, with a selection of curves going through point $(0, 0, 0)$. Each curve is confined to a normal plane and thus has an associated curvature as defined for the two dimensional case.

Gaussian curvature is an intrinsic measure in that it only depends on how distances are measured on the surface, not on how the surface is embedded. Gaussian curvature indicates whether a surface is *developable* or not. Developable surfaces are flat in one or more directions, *i.e.* one or both of κ_1 or κ_2 are zero. These kinds of surfaces are important in many applications because they can be parameterized with zero distortion. Developable surfaces include the plane, the cone and the tube.

Mean curvature on the other hand is an extrinsic measure and defined as the average of the principal curvatures

$$K_M = \frac{\kappa_1 + \kappa_2}{2}. \tag{2.7}$$

It can also be interpreted as the rate of change of surface area under small deformations in the normal direction. Mean curvature has certain properties that make it interesting in geometric modeling. A minimal surface is a surface which has mean curvature equal to zero everywhere. As a result minimizing mean curvature is equivalent to minimizing surface area and can be used as a smoothing operation.

Now, knowing formulas for computing mean and Gaussian curvature from principal curvatures might be enlightening, but from a practical viewpoint it is often difficult to find κ_1 and κ_2 for surfaces that have no inherent parameterization. However, if a parameterization is known the principle curvatures – and more – can be found using the fundamental forms of differential geometry. This will be discussed in Chapters 3 and 7.

We will also come back to non parameterized curves and surfaces in the next chapter (3) and derive curvature expressions for both implicit and explicit settings.

2.5 RIEMANNIAN MANIFOLDS

A *Riemannian manifold* is a differentiable manifold that has the additional structure of a *metric tensor*[2], or an inner product, defined for all tangent spaces. The metric tensor

2. Sometimes referred to as only metric.

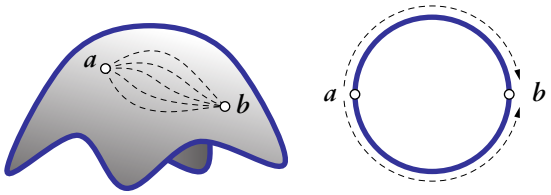


Figure 2.9: Geodesic distance in a manifold is the minimum of the length of all curves connecting the two points of interest. (a) Some of the curves connecting two points on a surface. Geodesic distance is not necessarily unique, in (b) there are two shortest paths.

g is smoothly varying function on the form $\langle \cdot, \cdot \rangle : T_{\boldsymbol{p}}M \times T_{\boldsymbol{p}}M \longrightarrow \mathbb{R}$ that generalizes the scalar product in Euclidean space to manifolds.

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^T g_{ij} \boldsymbol{v} \tag{2.8}$$

The metric tensor is a positive definite quadratic form and in this text we let the coefficients of g_{ij} be represented by the matrix $\boldsymbol{G}$,

$$\boldsymbol{G} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1d} \\ g_{21} & g_{22} & & \\ \vdots & & \ddots & \\ g_{d1} & & & g_{dd} \end{bmatrix} \tag{2.9}$$

giving

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^T \boldsymbol{G} \boldsymbol{v}. \tag{2.10}$$

Above, the position dependence of the metric tensor, $\boldsymbol{G} = \boldsymbol{G}(\boldsymbol{x})$, is implicit but should not be forgotten. Any space with an inner product has a naturally defined norm on the form

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}. \tag{2.11}$$

In fact Euclidean space does have a metric which is the identity matrix. This is most often ignored since $\boldsymbol{u}^T \boldsymbol{I} \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v}$.

Having an inner product and a norm defined on the tangent space of the manifold in a smoothly varying manner makes it possible to measure distances, angles, area, and more in the manifold.

Again foregoing matters slightly, we note that if a parameterization $\boldsymbol{f}$ is known, the Jacobian of the mapping (J_f) defines the metric of this parameterization domain.

$$J_f = \langle \nabla \boldsymbol{f}, \nabla \boldsymbol{f} \rangle . \tag{2.12}$$

$$J_f^T J_f = \boldsymbol{G} \tag{2.13}$$

The metric is sometimes also called the first fundamental form. We can also write it directly, in terms of the partial derivatives of the mapping:

$$\boldsymbol{G} = \begin{bmatrix} g_{11} & g_{12} & \cdots \\ g_{21} & \ddots & \\ \vdots & & \end{bmatrix}, \qquad g_{ij} = \frac{\partial \boldsymbol{f}}{\partial x_i} \cdot \frac{\partial \boldsymbol{f}}{\partial x_j}. \tag{2.14}$$

In this thesis we discuss geodesic distance in manifolds with applications in parameterization, interpolation, and path finding.

2.5.1 *Geodesic distance*

Distance is an important property of geometry. Knowing the pairwise distances between all points in manifold makes it possible to deduce much of its shape. Let us first study the length of an arbitrary parameterized curve connecting two points in a manifold, see Figure 2.9(a). The length l of a curve $c : [a, b] \longrightarrow M$ with $c(a) = \boldsymbol{a}$, and $c(b) = \boldsymbol{b}$ is

$$l_c = \int_c ds = \int_a^b \|\dot{\boldsymbol{c}}(t)\| \, dt = \int_a^b \sqrt{\dot{\boldsymbol{c}}(t)^T \boldsymbol{G}(\boldsymbol{c}(t)) \, \dot{\boldsymbol{c}}(t)} \, dt. \tag{2.15}$$

for any C^1 path in the manifold going from $\boldsymbol{a}$ to $\boldsymbol{b}$, independent of parameterization. Geodesic, or shortest, distance is then defined as the minimum length over all these curves

$$d(\boldsymbol{a}, \boldsymbol{b}) = \min \; l_c. \tag{2.16}$$

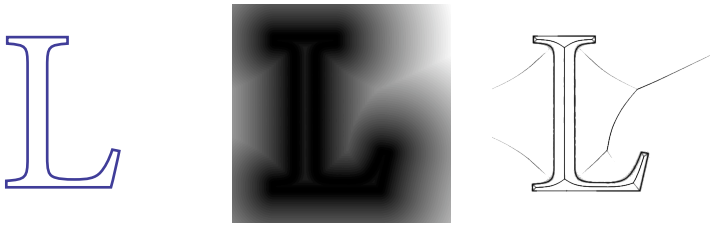
The geodesic distance measure is not necessarily unique, in Figure 2.9(b) we illustrate this with antipodal points on a circle. The set of points $\boldsymbol{B}$ for which there are more than one equal shortest path in $d(\boldsymbol{x}, \boldsymbol{S})$ define the *cut locus* or *medial axis* of $\boldsymbol{S}$. It is worth noting that the distance function of $\boldsymbol{S}$, $d(\boldsymbol{x}, \boldsymbol{S})$, is smooth except on the cut locus, see Figure 2.10.

We will come back to some of the properties of the Riemannian manifold in Chapter 6, for now we conclude that almost all of the geometry in this thesis is modeled as being differentiable, and sometimes also Riemannian.

2.6 SUMMARY

In this chapter we have formalized the concept of surfaces (and n-dimensional geometry) into manifolds, and also discussed topological issues and orientability. General criteria for when geometry can be classified as a manifold were found and several examples inspected. The introduction of manifolds and differential geometry was rather brief, for a complete reference to manifolds, and the differential aspects thereof, we refer the reader to [24, 42].

Some geometric properties of interest were derived for two- and three dimensional geometry. More specifically, we looked into the notion of curvature for higher dimen-



(a) (b) (c)

Figure 2.10: (a) The letter 'L' from the typeface Minion Pro. The curve defines the point set $\boldsymbol{S}$ for which $d(\boldsymbol{x}, \boldsymbol{S}) = 0$. (b) The distance function, $d(\boldsymbol{x}, \boldsymbol{S})$, visualized with black to white. (c) Discontinuities in d are located at the object outline, but also at its medial axis. This is indicated here through the norm of the gradient of the distance function, *i.e.* black regions have vanishing gradient.

sions than two and found measures useful for both parameterization and geometric smoothing.

CHAPTER 3

IMPLICIT AND EXPLICIT GEOMETRY

In this chapter we give a brief introduction to implicit and explicit geometry. Our main focus is surfaces – that is two dimensional geometry in a three dimensional space. The concept, nevertheless, generalizes to any dimension and curves and volumes are also mentioned. More emphasis is placed on the implicit representation as it pertains to the majority of thesis.

In order to help the reader main ideas are exemplified and illustrated throughout the text. We also rely on some of the necessary mathematical concepts that were introduced in the previous chapter. In particular we model geometry as *manifolds* – a generalization of normal space to a "curved" counterpart.

3.1 INTRODUCTION

Implicit geometry is, as the name suggests, an indirect specification. With an explicit representation a relationship between the spatial variables is established. A canonical explicit example is the equation for a line

$$y(x) = kx + m,$$

where the pair $x, y(x)$ generates points on the line. Given a general smooth function $y(x)$ it is obvious that the connectivity of the line is completely governed by the independent variable x.

For implicit geometry, on the other hand, both the location of the surface and its connectivity are defined by points in space that satisfy some requirement. This is often specified as a mathematical function $f(\boldsymbol{x})$. The geometry is then defined as the points for which the function evaluates to zero. In this manner the function f through the equation

$$f(\boldsymbol{x}) = 0 \tag{3.1}$$

implies the existence of the geometry rather than explicitly point out its location. We now examine explicit geometry and see how it is inherently parameterized before we move on to the definition of implicit geometry.

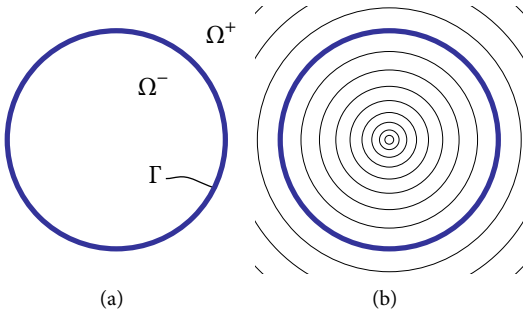


Figure 3.1: (a) The unit circle, $f(x, y) = x^2 + y^2 - 1 = 0$, partitions the plane into an inside, Ω^-, and an outside, Ω^+. The separating layer (Γ) is the level-curve for $f(x, y) = 0$. (b) Some of the level-curves for $f(x, y) = c$.

3.2 PARAMETRIC GEOMETRY

Let us start by looking at a typical example, the equation for the unit circle, see Figure 3.1(a):

$$x^2 + y^2 = 1. \tag{3.2}$$

An explicit version of the equation couples x and $y(x)$ like so

$$y(x) = \pm\sqrt{1 - x^2}, \quad x \in [-1, 1], \tag{3.3}$$

and thus maps two values for each valid x onto y. This gives a collection of x and y for which the pair is always a position on the circle, and as such the equation is explicit. However, having two y-values for each x is not optimal from a mathematical point of view. Better yet is to use trigonometric functions to define the circle. This gives a one-to-one (injective) mapping:

$$\boldsymbol{f}(t) = \{x(t), y(t)\} = \{\cos(t), \sin(t)\}, \quad t \in [0, 2\pi). \tag{3.4}$$

This expression introduces the independent variable t, a *parameter*, which made it possible to decouple the spatial variables x, y from each other. We can see this as a one-dimensional vector-valued *parametrization* on the form

$$\boldsymbol{f} : \Omega \subset \mathbb{R} \longrightarrow \mathbb{R}^2, \tag{3.5}$$

with the requirement that $\boldsymbol{f} = (f_1, f_2)$ is continuous and injective. Explicit geometry is often called parametric due to its inherent parametrization. In mathematical jargon we say that Equation (3.4) defines a parametric curve C as the image of Ω under the parametrization:

$$C = \boldsymbol{f}(\Omega) = \{\boldsymbol{f}(t) \mid t \in \Omega\}. \tag{3.6}$$

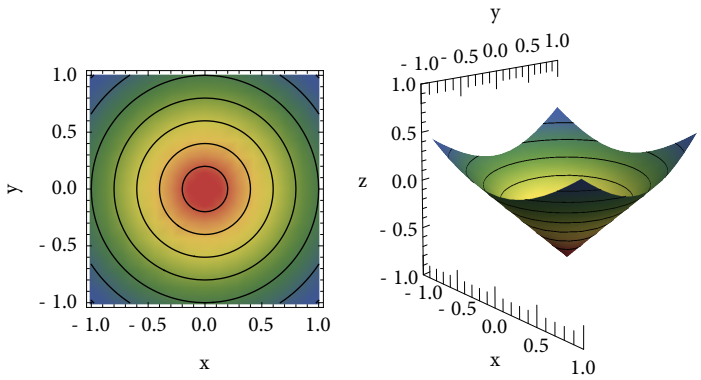


Figure 3.2: (a) The function $f(x, y) = x^2 + y^2 - 1$ color coded, with iso-contours overlayed. The graph of the same function $f(x, y) = z$.

We note that the curve C is completely characterized by the parameterization function $\boldsymbol{f}$. Consider, for example, the parameterization in Equation (3.4) defining the circle. It is an analytic function and it follows that changing the variable t smoothly over the parameter domain Ω automatically gives a smooth curve in $\mathbb{R}^2$ as well. This powerful and important feature extends to all dimensions.

It is clear that a parameterization is not necessarily unique. Both Equation (3.3) and (3.4) give positions on the unit circle but only Equation (3.4) has a defined inverse $\boldsymbol{f}^{-1}$. Its parameterization is *bijective*.

3.3 IMPLICIT GEOMETRY

We just showed how explicit geometry is generated as pairs or *tuples* from a vector-valued function describing coordinates in space. The tuples always lie on the geometry. Consider, on the other hand, the implicit equation for the unit circle

$$f(x, y) = x^2 + y^2 - 1. \tag{3.7}$$

This form, except for being real valued, is without restrictions on any of the variables, $(x, y) \in \Omega \subseteq \mathbb{R}^2$. The function may be evaluated wherever in the plane, assigning a scalar to each query point. The values of these scalars encode the circle by giving points outside a positive sign, and inversely points inside a negative sign. These partitions are referred to as Ω^+, and Ω^-, as seen in Figure 3.1. Furthermore, by choosing the points where $f(x, y) = 0$, a subset of the plane is selected which is equivalent to the circle. Such a subset is called a *level-curve*, or more generally *level-set*, of the implicit function.

Alternate names are *iso-surface*[1], *level-surface*, *locus of zeros*, and the dimensionless *interface*. We denote the level-set with Γ.

The level-set is an object of *co-dimension* one, meaning that it has the same dimensionality as the scalar function minus one. Thus, in the case of the circle above, the geometry is a one-dimensional curve and the scalar function is evaluated in the plane. Subsequently a surface can be defined by a three-dimensional scalar function, a volume by a four-dimensional scalar function, and so forth. The function space is also referred to as the *embedding* — the circle is embedded in the plane.

If an n-dimensional level-set is closed and bounded, the inside is an $(n+1)$-dimensional "flat" geometry. That is, the inside of a closed curve in the plane defines a flat surface bounded by the curve. For surfaces this is particularly convenient since volumes can be represented by closed surfaces embedded in $\mathbb{R}^3$, instead of as volumes embedded in (costly) $\mathbb{R}^4$.

Definition

Let $\boldsymbol{x}$ denote positions in an n dimensional space, which for most of this thesis will be 2D or 3D, but for the sake of generality we define a general scalar function to be

$$f : \Omega \subset \mathbb{R}^n \longrightarrow \mathbb{R}. \tag{3.8}$$

Now the implicit geometry Γ can be formally defined as the *preimage* (inverse image) of the iso-value under f

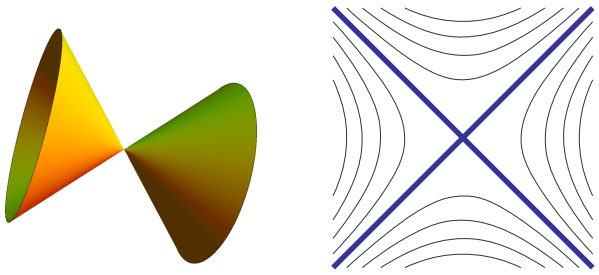
$$\Gamma \equiv f^{-1}(c) = \{\boldsymbol{x} \in \Omega \mid f(\boldsymbol{x}) = c\}, \tag{3.9}$$

which is the set of points $\boldsymbol{x}$ for which $f(\boldsymbol{x})$ evaluates to c. Changing the constant c, called the *iso-value*, allows for the selection of different level-sets, see Figure 3.1 for a 2D example. Most often the iso-value is set to zero, which is convenient since the sign of f then can be used for inside/outside queries. In this thesis, if not stated otherwise, we will assume that the scalar function is transformed such that the iso-value becomes zero.

A closed implicit manifold is a co-dimension one object that separates the embedding into an inside and outside region and hence the name interface. The definition of whether $f < 0$ should denote inside or outside is, of course, optional. We choose $f < 0 \equiv$ inside because then the gradient will point in the same direction as the surface normal.

Storing geometry using a higher dimension than necessary can seem wasteful at first. But, using the scalar function, we can categorize *the full domain* Ω because the interface is embedded in a higher dimensional space. This gives the opportunity to query points that are *not on the interface*, which is not possible without considerable effort, using explicit representations.

1. The prefix iso is a Greek word meaning equal.



(a) (b)

Figure 3.3: (a) Two cones plotted as the zero level-set of $f(x, y, z) = -x^2 + y^2 + z^2$. The apex of the cones is a singular point since $\nabla f = \varnothing$. (b) Some level curves of $f(x, y, 0)$. The curves reveal that f is smooth and continuous for iso-values other than 0.

3.3.1 *The implicit function theorem and smoothness*

So far no restrictions have been placed on the implicit function. We have seen though, that some implicit functions can generate non-manifolds, *e.g.* the zero-set of the cone function. Much of the differential geometry toolbox developed in the last chapter applies only to manifolds. Therefore, we want a way of determine whether an implicit function generates manifolds or not.

The classification can be done by examining the smoothness of f, and in this section we will see that it is required to be *continuous* and *differentiable* to produce manifolds.

The surface gradient is only defined for points for which the partial derivatives $\partial f/\partial x_i$ are continuous and not all zero. We let the existence of the gradient embody our requirements. This is then used to classify points as *regular* (existing gradient) or *singular* (vanishing gradient). If an iso-value exclusively generates regular points it is said to be a *regular value*, that is

$$c \text{ is a regular value if } \nabla f(\boldsymbol{x}) \neq \varnothing, \ \forall \boldsymbol{x} \in f^{-1}(c). \tag{3.10}$$

The *implicit function theorem*[14] states that if c is a regular value of f, then $f^{-1}(c)$ is a manifold. More specifically, the theorem says that each regular point defines a local manifold geometry [6]:

> *Given an implicit function, $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, there exists a local neighborhood $\boldsymbol{N}$ of $\boldsymbol{p} \in \mathbb{R}^n$ such that $f^{-1}(c) \cap \boldsymbol{N}$ is a parameterized $(n-1)$-manifold if $\boldsymbol{p} \in f^{-1}(c)$ is a regular point.*

For surfaces this asserts that any two-dimensional geometry described by an implicit function in $\mathbb{R}^3$ is a two-dimensional[2] manifold for regular iso-values. In the same

2. $3 - 1$

fashion manifold curves and volumes (in their respective embeddings) are guaranteed by regular iso-values.

We now use the classification to investigate some of the earlier examples. The cones in Figure 3.3 and 2.3(d), for example, have a vanishing gradient and are therefore not regular. Thus the geometry is not a manifold. At other iso-values the gradient is continuous and the geometry is therefore a manifold. This agrees with the Euclidean space homeomorphy conditions that we reviewed in the previous chapter.

We conclude by re-stating the most important fact of this section. If the gradient of the implicit function is continuous (and not zero) over the level-set we are interested in, then the implicit theorem guarantees that this level-set is a manifold.

3.4 DIFFERENTIAL PROPERTIES

The true strength of the implicit representation comes into light when one tries to derive differential quantities for a manifold. Depending on the order of the differential of interest, the requirements on continuity (the partial derivatives) vary. From the definition of a differentiable manifold we know that our surfaces are at least C^1. This was also confirmed for implicit geometry through the implicit function theorem.

The first (and sometimes forgotten) differential quantity is the zeroth partial derivative, which for implicit geometry is the scalar function itself. In this case the zeroth differential often describes proximity to the surface. This distance measure is seldom Euclidean though. Equation (3.7), for example, measures the squared distance to the closest point on the circle. For explicit geometry it is not meaningful to talk about distance since the parameterization only samples the surface, not the embedding.

3.4.1 Normal and tangent vectors

The foremost differential quantity in computer graphics is surely the surface normal, $\vec{n}$. It was introduced using the existence of tangent spaces in the last chapter, see Figure 2.6. For a parametric surface that is differentiable, the partial derivatives,

$$\boldsymbol{f}_u = \frac{\partial \boldsymbol{f}}{\partial u} \qquad \text{and} \qquad \boldsymbol{f}_v = \frac{\partial \boldsymbol{f}}{\partial v}$$

are tangential to the surface by construction. Assuming a regular parametrization, that is $\boldsymbol{f}_u \times \boldsymbol{f}_v \neq 0$ the vectors span the local tangent plane $T_{\boldsymbol{p}}M$. Therefore we find the surface normal vector as

$$\vec{n} = \frac{\boldsymbol{f}_u \times \boldsymbol{f}_v}{\|\boldsymbol{f}_u \times \boldsymbol{f}_v\|}. \tag{3.11}$$

The surface normal for an implicit surface can be derived using the *gradient* (∇), an operator that gives the direction and magnitude of the steepest ascent of the function operated on. Because it is orthogonal to the level-sets of any function by construction it defines the surface normal:

$$\vec{n} = \frac{\nabla f}{\|\nabla f\|}. \tag{3.12}$$

In this work we chose the scalar function such that the inside of our implicit geometry has negative sign. Then the normal and the gradient point in the same direction (outwards). Otherwise the direction of the normal as computed above in Equation (3.12) should be reversed.

3.4.2 Curvature

Mathematically, curvature κ is defined as the norm of the second order derivative for a curve with respect to its arc-length, or natural, parametrization

$$\kappa(s) = \|\ddot{\boldsymbol{c}}(s)\|. \tag{3.13}$$

The natural parametrization can be difficult to find though. Then the following formula

$$\kappa(t) = \frac{|\dot{x}(t)\,\ddot{y}(t) - \dot{y}(t)\,\ddot{x}(t)|}{\left(\dot{x}(t)^2 + \dot{y}(t)^2\right)^{3/2}} \tag{3.14}$$

provides a way of computing the curvature from an arbitrarily parameterized curve $\boldsymbol{c}(t) = \{x(t), y(t)\}$.

Implicit curvature formulas

For implicit geometry there is no inherent parametrization and the above formulas are not applicable. Often in the literature, curvature for implicit geometry is defined as the divergence of the normal [24, 54]

$$\begin{aligned} f &: \mathbb{R}^2 \longrightarrow \mathbb{R} \\ \kappa &= \nabla \cdot \left(\frac{\nabla f}{\|\nabla f\|} \right). \end{aligned} \tag{3.15}$$

Due to the exhaustive expansion of this expression alternate formulas are sometimes more practical [34].

Curvature in higher dimensions

For a parametric surface the first fundamental form $\boldsymbol{I}_f$ of the parametrization $\boldsymbol{f}$ is

$$\boldsymbol{I}_f = \begin{pmatrix} \boldsymbol{f}_u \cdot \boldsymbol{f}_u & \boldsymbol{f}_u \cdot \boldsymbol{f}_v \\ \boldsymbol{f}_v \cdot \boldsymbol{f}_u & \boldsymbol{f}_v \cdot \boldsymbol{f}_v \end{pmatrix} = \begin{pmatrix} E & F \\ F & G \end{pmatrix}. \tag{3.16}$$

It is, like the metric, a positive definite quadratic form that defines an inner product on $T_{\boldsymbol{p}}M$. We have already explained that curvature is a second order effect and the existence of second order derivatives is therefore assumed. The second fundamental form is

$$\boldsymbol{II}_f = \begin{pmatrix} \boldsymbol{f}_{uu} \cdot \vec{n} & \boldsymbol{f}_{uv} \cdot \vec{n} \\ \boldsymbol{f}_{vu} \cdot \vec{n} & \boldsymbol{f}_{vv} \cdot \vec{n} \end{pmatrix} = \begin{pmatrix} L & M \\ M & N \end{pmatrix}. \tag{3.17}$$

The product $\boldsymbol{I}_f^{-1}\boldsymbol{II}_f$ defines the *shape operator*[24], also called the Weingarten map or second fundamental tensor

$$\boldsymbol{S} = \boldsymbol{I}_f^{-1}\boldsymbol{II}_f = \frac{1}{(EG - F^2)}\begin{pmatrix} FM - GL & FN - GM \\ FL - EM & FM - EN \end{pmatrix}. \tag{3.18}$$

The shape operator is a measure of the directional derivative of the normal such that $\boldsymbol{S}\vec{t} = \frac{\partial \vec{n}}{\partial \vec{t}}$. The shape operator can also be defined in terms of its eigenvalues (the principal curvatures) and eigenvectors (the principal directions) as

$$\boldsymbol{S} = \begin{pmatrix} \vec{t}_1 & \vec{t}_2 \end{pmatrix}\begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}\begin{pmatrix} \vec{t}_1 & \vec{t}_2 \end{pmatrix}^{-1}. \tag{3.19}$$

Due to the invariance of the determinant and trace operator under similarity transforms[3] we can measure Gaussian and mean curvature directly as

$$K_G = \det(\boldsymbol{S}) = \det(\boldsymbol{I}_f^{-1}\boldsymbol{II}_f) = \frac{\det(\boldsymbol{II}_f)}{\det(\boldsymbol{I}_f)} = \frac{LN - M^2}{EG - F^2}, \tag{3.20}$$

and

$$K_M = \frac{1}{2}\mathrm{trace}\,(\boldsymbol{S}) = \frac{1}{2}\mathrm{trace}\left(\boldsymbol{I}_f^{-1}\boldsymbol{II}_f\right) = \frac{LG - 2MF + NE}{2(EG - F^2)}. \tag{3.21}$$

The relationship between these curvature formulas and the principal curvatures (Equations (2.6 and 2.7)) also give the opportunity to find κ_1 and κ_2 without computing eigenvalues.

Implicit surfaces

For implicit surfaces we must find formulas that do not rely on a parameterization. Resorting to [24, 34, 54] we calculate Gaussian and mean curvature directly in an implicit setting. The divergence formula in Equation (3.15) is immediately applicable for surfaces, except for a dimension coefficient of 1/2, and here it measures mean curvature.

$$\begin{aligned} f &: \mathbb{R}^3 \longrightarrow \mathbb{R} \\ K_M &= \frac{1}{2}\nabla \cdot \left(\frac{\nabla f}{\|\nabla f\|}\right). \end{aligned} \tag{3.22}$$

See [34] for more formulas, including methods to compute the Gaussian curvature.

It turns out that the shape operator in has a parameterization independent analogy, the *shape tensor* $\boldsymbol{T}$. It is defined as the three by three matrix

$$\boldsymbol{T} = \begin{pmatrix} \vec{t}_1 & \vec{t}_2 & \vec{n} \end{pmatrix}\begin{pmatrix} \kappa_1 & 0 & 0 \\ 0 & \kappa_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} \vec{t}_1 & \vec{t}_2 & \vec{n} \end{pmatrix}^{-1}. \tag{3.23}$$

3. Both the determinant and trace operator obeys $\mathrm{op}(\boldsymbol{A}) = \mathrm{op}(\boldsymbol{BAB}^{-1})$.

The eigenvalues and eigenvectors of $\boldsymbol{T}$ give, like for the shape operator, the principal curvatures and directions. From the definition in Equation (3.23) we learn that in addition $\boldsymbol{T}$ has the third eigenvalue 0 and the third eigenvector $\vec{n}$. In [54] a method to find the shape tensor for an implicit surface was described. The procedure first computes the directional derivative of the normal in the embedding space and then projects the result down on the tangent plane $T_p M$.

3.5 SUMMARY AND FURTHER READING

In this chapter we introduced the implicit and explicit geometry representations and pointed out some of their benefits and drawbacks. A specific case using the implicit function theorem was also given that simplifies classification based on the existence of the gradient.

Differential aspects such as gradients, normals, and curvature were derived for two- and three dimensional implicit and parametric geometry.

For the interested reader, a good introduction to implicit surfaces is given in [6]. Most general text books on computer graphics contain sections on implicit geometry, *e.g.* [30], which also has plenty of information on explicit geometry. Some of the differential equations are taken from [34], which is an exhaustive source of curvature formulas for implicit geometry.

CHAPTER 4

LEVEL-SET METHODS

This chapter introduces one of the core concepts of this thesis — the use of a grid based dynamic implicit geometry. We begin by giving some background and motivation. Then, we dive straight into the theory and show how to "drive" these dynamic surfaces. The level-set method is a numerical method; this has many practical implications. We discuss some of the more important ones in this chapter and show how they relate to the implicit setting. The chapter is finally concluded with a hands-on example.

4.1 INTRODUCTION

The level-set method, or LSM for short, is a grid based version of implicit geometry equipped with an associated toolbox of dynamic operations. LSM was introduced in 1988 by Osher and Sethian [66] as a tool for the tracking of propagating interfaces in time dependent physics problems. Since then, the level-set method has been applied to an abundance of problems in physics, chemistry, fluid mechanics, image processing, computer vision, computer graphics, and more.

As the method has developed, the toolbox has been refined and extended. Much of the effort has been devoted to creating numerical methods that are accurate and robust. Some of the relevant research will be reviewed in this chapter.

Because LSM uses an implicit representation the geometry is embedded in a higher dimensional space. For detailed geometry this can be a serious impediment when using LSM on a computer with limited resources. This has put a strong focus on the computer science aspects of LSM, such as algorithm and implementation details. Several approaches have been proposed to deal with efficiency issues, including work that forms part of this thesis. This chapter introduces some of the theoretical aspects, such as discretizations in time and space. In the following chapter (5) we will focus on the practical issues of LSM.

4.1.1 Motivation

The level-set method simplifies topology issues gracefully due to its implicit representation as discussed in the previous chapter. Topological ambiguities and self-intersections are impossible, a point in space can only be inside or outside, as it is completely classified by the single value of a scalar function. This, and other benefits, have made LSM the preferred alternative for the tracking of free surfaces in fluid simulations. Figure 4.1 illustrates the difficulties that can arise when two stylized waves collide in an explicit

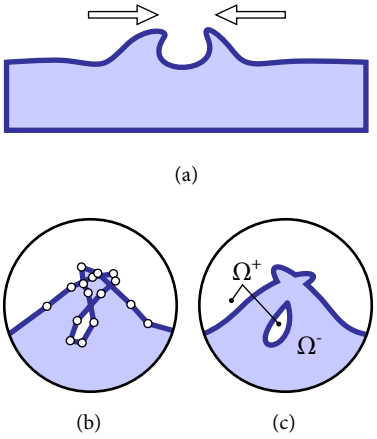


Figure 4.1: (a) An illustration of topology issues for "colliding" water. (b) With an explicit representation it can be difficult to determine topology due to self intersections. (c) An implicit representation resolves this automatically – there is only one interpretation of inside/outside and hence no self intersections.

and implicit setting. The extreme behaviour, from a topological point of view, makes explicit geometry an unattractive alternative. Furthermore an explicit, or Lagrangian, sampling is susceptible to aliasing artifacts and instabilities due to distortion already at "well behaved" deformations of the interface. Proposed solutions to both aliasing and topological problems in front-tracking exist: re-meshing, filtering, smoothing (see *e.g.* [11, 65, 75, 87]), but these remedies are quite complex and introduce non-physical behaviour. For other alternatives to LSM, one should also mention closed form implicit geometry, but finding analytic expressions for complex shapes is very difficult, and will not be considered here.

4.2 THE LEVEL-SET METHOD

4.2.1 Definitions

As previously stated the level-set method is a sampled implicit geometry representation. In this text the symbol ϕ denotes the implicit function when used in LSM. As before, the interface is denoted by Γ and given by the preimage of the iso-value c under ϕ. Equation (3.9) is restated here for convenience:

$$\Gamma \equiv \phi^{-1}(c) = \{\boldsymbol{x} \in \mathbb{R}^n \mid \phi(\boldsymbol{x}) = c\}. \tag{4.1}$$

In order to introduce dynamics for an interface described by Equation (4.1) we need to be able to define the motion of any point in the embedding. The "engine" of

the level-set method can be given in two principal forms. The first one is achieved by letting the motion of each point in the embedding be represented by an external vector field $\vec{V}$, that is

$$\frac{d\boldsymbol{x}}{dt} = \vec{V}(\boldsymbol{x}). \tag{4.2}$$

This can be seen as passive *advection*. Each point on the interface just moves along with the vector field. The other form for driving the motion of the interface is based on the surface normal. In the previous chapter it was shown that the normal of any level-set of ϕ is

$$\vec{n} = \pm\frac{\nabla\phi}{\|\nabla\phi\|}. \tag{4.3}$$

We choose the positive solution based on the assumption that Ω^- has negative sign. This convention makes the gradient and the normal point in the same direction. Equation (4.3) can then be used to define the scalar valued level-set *speed function*, which is

$$F(\boldsymbol{x}, \vec{n}, \phi, \dots) = \vec{n} \cdot \frac{d\boldsymbol{x}}{dt} = \frac{\nabla\phi}{\|\nabla\phi\|} \cdot \frac{d\boldsymbol{x}}{dt}. \tag{4.4}$$

It is the speed of a point, $d\boldsymbol{x}/dt$, projected on the normal vector $\vec{n}$ of the level-set function at $\boldsymbol{x}$. The speed function effectively describes the motion of the interface in the normal direction and can be used to evolve the level-set over time. Arguments to F include the level-set function itself. This is to signify that the speed at a point $\boldsymbol{x}$ can depend on inherent properties of the interface, such as the normal, or the curvature. We also see that Equation (4.4) can be transformed into a pure vector form (4.2) by considering the normal vector as a direction with the speed function as magnitude.

To arrive at a dynamic interface we derive equations of motion by introducing time dependence into Equation (4.1). This can be done in two different fashions. The first version varies the iso-value over time, such that

$$\Gamma(t) = \{\boldsymbol{x}(t) \in \mathbb{R}^n \mid \phi(\boldsymbol{x}(t)) = c(t)\}. \tag{4.5}$$

This describes the evolution of the interface of an implicit function as the iso-value changes and is called the *static level-set formulation*. In Figure 3.1 some level-curves of a unit circle are drawn; the static formulation allows the interface to smoothly transition between these level-curves as a function of time. But, because the level-sets cannot intersect by definition, the range of motion becomes quite limited. This formulation is fast and straightforward to compute with LSM, and the ability to produce off-set surfaces is valuable.

The second dynamic formulation instead introduces time-dependence for the level-set function itself:

$$\Gamma(t) = \{\boldsymbol{x}(t) \in \mathbb{R}^n \mid \phi(\boldsymbol{x}(t), t) = c\}. \tag{4.6}$$

This is accomplished by differentiating Equation (4.6) with respect to time

$$\frac{d}{dt}\left[\phi\left(\boldsymbol{x}\left(t\right),t\right)\right] = \nabla\phi \cdot \frac{d\boldsymbol{x}}{dt} + \frac{\partial\phi}{\partial t},$$

which results in an advection equation. We then rearrange and substitute the definition for the speed function in Equation (4.4)

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \cdot \vec{V} \tag{4.7}$$

$$= -\left\|\nabla\phi\right\| F\left(\boldsymbol{x}, \vec{n}, \phi, \dots\right). \tag{4.8}$$

These partial differential equations (PDEs) give rise to the dynamic level-set formulation and are referred to as the *fundamental level-set equations*. Equation (4.7) is also known as the G-equation in combustion theory [50].

How are Equations (4.7 and 4.8) to be interpreted? The equations are clearly algebraically equal. They describe the same connection, namely how the interface moves over time. From a physical point of view, however, they are quite different, as we will see in this chapter.

Equation (4.7) describes the transportation of the interface in an external vector field. This is called advection. One example is free surface simulation where an interface is advected in a flow field given by, for example, the Navier-Stokes equations. Equation (4.8), on the other hand, can be interpreted as the motion of the interface in its normal direction by a magnitude determined by the speed function. This follows from the fact that the projection of the gradient on the vector field ($\nabla\phi \cdot \vec{V}$) discards the vector components that are orthogonal to the surface normal. The level-set equation on form (4.8) lends itself naturally to propagation where the motion depends on the geometry itself. Surface smoothing, for example, moves the interface such that its area is minimized. This effectively removes sharp features while keeping low frequency information. The smoothing can be achieved by employing a speed function depending on local mean curvature $F = -\alpha\kappa$, with $\alpha > 0$ [19].

From the form of the fundamental level-set equations we see that the range of motion is completely unrestricted at the cost of having the solutions exist in a higher dimensional space. The dynamic level-set equations are the preferred choices for most applications in complex interface capturing.

4.2.2 Sampled geometry

So far the implicit geometry description has been continuous. Now we introduce the *grid*. It is presented in two dimensions for clarity. The extension to higher (or lower) dimensions is straightforward. Let the level-set function ϕ be sampled on a fixed rectilinear grid as shown in Figure 4.2, and 4.3.

We start with some preliminaries and notation. Let Δx and Δy denote the grid spacing on the rectilinear grid. Most often the spacing is uniform, so that $\Delta x = \Delta y = h$. We use this to form a terse subscript notation that labels the sampled value at

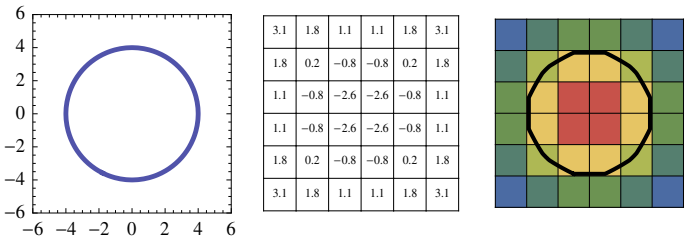


Figure 4.2: With LSM, the implicit function is discretely sampled on a grid. (a) The zero level-curve for the function $x^2 + y^2 = 4$. (b) The discrete (truncated) samples on the grid. (c) The reconstructed zero level-curve overlayed on the grid. For practical reasons we use a color coding (red-blue) when the individual numbers are of less importance.

integer position i, j in the grid with ϕ_{ij}. Equivalently, the more descriptive expression $\phi(i\Delta x, j\Delta y)$ can be used. For clarity the sub- and superscripts are sometimes dropped when obvious or irrelevant.

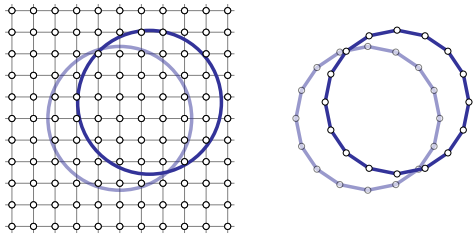
The interface is defined as the zero-set of the level-set function. Generally, the interface, or any arbitrary point of interest, will not lie exactly on the lattice nodes. In order to find the level-set function at any point in space we need to reconstruct a continuous function from the sampled values. As seen in Figure 4.2(c) this issue needs to be handled carefully in order to accurately capture the interface, or any level-set, on the grid.

The fundamental level-set equations introduce time dependence in order to represent dynamic geometry. Let the time samples be denoted ϕ^n, where n is identifying the time $n\Delta t$. The complete notation for the level-set function now reads

$$\phi^n_{ij} = \phi\left(i\Delta x, j\Delta y, n\Delta t\right). \tag{4.9}$$

Sometimes, when the spatial position is beside the point, we will only use the time index, $\phi^0 = \phi\left(t_0\right)$.

It is important to understand how the time dependence affects the implicit grid-based geometry. In Figure 4.3(a) it is illustrated how the grid remains fixed even though the geometry is shifting. The movement is achieved by changing the level-set function over time, *i.e.* changing the values of the grid instead of moving the grid itself. This is possible because the level-set function is defined over the embedding space. This stands in contrast to an explicit representation, see Figure 4.3(b), where the samples themselves move over time. In this form only the actual geometry is sampled, not its embedding. The explicit view is sometimes called *Lagrangian* since the sampling follows the geometry. In physics, a Lagrangian frame is following individual particles as they move through space. To exemplify we make an analogy with a river as a flow of some quantity that we are interested in measuring. The Lagrangian view is an observer



(a) (b)

Figure 4.3: Capturing geometry as it moves over time. Light blue is time step t_0, dark blue is t_1. (a) The implicit setting: Even though the circle is moving, the grid remains fixed. (b) An explicit grid moves with the geometry.

drifting down the river in a boat. The implicit view, on the other hand, is referred to as *Eularian* and the sampling is fixed in space. In the river analogy this is a stationary observer on the river bank inspecting the water as it runs by.

4.2.3 *The level-set function*

For implicit geometry it is insignificant what type of function ϕ is as long as it defines the interface in a consistent manner and retains adequate continuity as dictated by the implicit function theorem [14]. In the previous chapter we saw examples of circles and cones, as well as more complicated geometry, with many different types of implicit functions. In the discrete case one special function has shown be to a good choice. The *signed distance function* measures the Euclidean distance from a point in space to the *closest* point on the interface. It additionally discerns between interior (Ω^-) and exterior points (Ω^+) by assigning them different signs. We want ϕ to be a functions such that

$$\phi(\boldsymbol{x}) = \begin{cases} d(\boldsymbol{x}, \Gamma), & \text{for } \boldsymbol{x} \in \Omega^+ \\ -d(\boldsymbol{x}, \Gamma), & \text{for } \boldsymbol{x} \in \Omega^-, \end{cases} \tag{4.10}$$

where $d(\boldsymbol{x}, S)$ measures the geodesic (closest) distance between $\boldsymbol{x}$ and the general shape S, which in this case is the interface.

The implicit function theorem [14] tells us that the level-set function is required to have continuous and well defined gradients everywhere, in order to guarantee a manifold geometry. For the discretely sampled level-set function this transforms into another requirement, namely that of *Lipschitz continuity*:

$$|\phi(\boldsymbol{x}_1) - \phi(\boldsymbol{x}_2)| \leq C\,|\boldsymbol{x}_1 - \boldsymbol{x}_2|\,. \tag{4.11}$$

This constraint puts a bound on the magnitude of the rate-of-change of the sampled level-set function. In this way the continuity requirement on the gradients of ϕ is relaxed but instead the rate-of-change must be bounded by a finite constant. In fact,

the signed distance function, from Equation (4.10), can be defined by a constraint on the level-set function's gradients:

$$\|\nabla\phi(\boldsymbol{x})\| = 1. \tag{4.12}$$

This formula is called the *eikonal equation*, and it is well known that its viscosity solution, see for example [5, 26], produces bounded Euclidean distance functions that have unit length gradient in smooth regions. Hence the signed distance function is a Lipschitz function. We will get back to the concept of viscosity solutions in Section 4.4.3.

A lower bound on the Lipschitz constant in Equation (4.11) relates to a well conditioned function, meaning that it is suitable for digital computation. This is also supported by the fact that subtracting numbers that differ greatly in magnitude on a computer is sensitive to round-off errors, see for example [33]. A bound on the gradients limits the effects of this when computing, for example, derivatives by finite differences. This reveals the signed distance function to be a prime candidate for the level-set implicit function from a digital point of view.

One of the advantages with any type of distance function is that proximity information is available all over the embedding. The properties of the signed distance function defined in Equation (4.10) also make it possible to find not only the closest distance to the interface but also its position. This is called the *closest point transform*. In Euclidean space, using the positive outside sign convention, the closest point transform is given by

$$\boldsymbol{x}_\Gamma = \boldsymbol{x} - \phi(\boldsymbol{x})\,\nabla\phi(\boldsymbol{x}). \tag{4.13}$$

A signed distance function facilitates many important operations in computer graphics. Proximity, position, and topology information is directly available when querying the embedding space. This is most useful for problem that deals with space partitioning, see [9] for a recent example of collision detection.

4.3 A DISCRETE TOOLBOX

The dynamic level-set formulation and the fundamental level-set equations can be used to move the interface by solving either of the PDEs over time [66]. In order to do this there are quite a few things to consider. First, we need to address the spatial and temporal discretization, that is find out how to compute $\nabla\phi$ and $\partial\phi/\partial t$. Then, there is the mathematical classification of the actual PDEs. It turns out that the proper discretizations are depending on the PDE-class, which introduces further complications. Finally, we need to look at under what circumstances the resulting discretizations are numerically stable.

4.3.1 Discrete derivatives with finite differences

In LSM most of the numerics is based on *finite differences* for the approximations to the differential operators in the PDEs. We begin by giving a brief introduction on how

to form the most basic finite differences in both space and time. This is done from first principles using Taylor's theorem on a rectilinear grid in order to give the reader some intuition for the subject. Then, we briefly describe some of the more advanced schemes created specifically for the level-set method and give references for further reading.

Taylors theorem gives a polynomial approximation to a differentiable function around a given point. As the number of polynomials in the series increases and/or the distance to the given point decreases the approximation error, $R(x)$, tends to zero. Taylor's theorem states that

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x). \quad (4.14)$$

Additionally, if the function approximated is $(n+1)$ times differentiable and has bounded derivatives, then the error term is $R_n(x) = O((x-a)^{n+1})$ using big O notation.[1]

We now use Taylor's theorem to attain approximations to the derivatives for the level-set implicit function ϕ. The first order one dimensional Taylor expansion to the point $x+h$ about x can be found by omitting all terms of second order and above from Equation (4.14):

$$\phi(x+h) = \phi(x) + \phi'(x)h + O(h^2). \quad (4.15)$$

By rearranging the terms an expression for the first derivative of ϕ at x can be singled out

$$\phi'(x) = \frac{\phi(x+h) - \phi(x)}{h} + O(h), \qquad h \neq 0. \quad (4.16)$$

Because the error term is linear in h we say that the approximation is first order accurate, or $O(h)$. In practice we omit the error term and only work with the approximation. A two-dimensional level-set function, $\phi(x, y)$, depends on both x, and y, so in order to compute derivatives we use *partial* derivatives $(\partial\phi/\partial x_i)$. This is done by fixing the index not under consideration. A discrete partial derivative in x is

$$\frac{\partial\phi}{\partial x} \approx \phi_x^+ = \frac{\phi_{i+1,j} - \phi_{i,j}}{h}. \quad (4.17)$$

The plus sign denotes a *forward difference* and the subscript single x shows it to be a first derivative in the x direction.

By substituting the expansion point, $(x+h)$, with $(x-h)$ in Equation (4.15) we, instead, arrive at a first order accurate *backward* difference formula

$$\frac{\partial\phi}{\partial x} \approx \phi_x^- = \frac{\phi_{i,j} - \phi_{i-1,j}}{h}. \quad (4.18)$$

1. See [46] for more information on the big O notation.

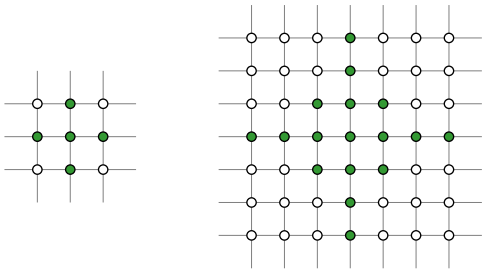


Figure 4.4: Some finite difference stencils. The green sample points are used for (a) a first order forward/ backward and second order central differences, (b) an accurate curvature measure, see [65].

A second order Taylor expansion producing a second order accurate *central* difference approximation can, similarly, be derived to be

$$\frac{\partial \phi}{\partial x} \approx \phi_x^0 = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h}. \tag{4.19}$$

The higher the order of the derivative is, and the higher the desired accuracy is, the more terms are needed in the finite difference approximation. We call the set of terms $\{\phi_{i,j}\}$ used to compute the approximation a *stencil*, see Figure 4.4. As seen in (b), a stencil can contain a large amount of grid points, some which are far away from the center point. This has implications for the implementation on a computer where memory architecture and access times can have large affects on the efficiency of the method.

Accurate spatial discretizations

First or second order discretizations are sufficient in some cases. But often, more accurate simulation results are required resulting in the need for higher order finite difference approximations. It is possible to derive such approximations using the Taylor expansion as shown above, by simply including more terms in the expansion. For smooth regions this is an attractive and straightforward solution. However, the nature of the distance field representing the interface often has, or develops, discontinuous regions (see for example Figure 4.7). Therefore, a number of tailor made finite difference schemes have been published that are developed especially for LSM.

In the presence of discontinuities the essentially non-oscillatory (ENO) [67] or weighted ENO (WENO) [48, 76, 77] polynomial interpolation techniques are well suited. They allow for an accurate representation of geometry with sharp features. The ENO scheme chooses the derivative that corresponds to the smoothest Newton polynomial approximation to ϕ on the interval found using divided differences. In this way oscilla-

tions from discontinuities are minimized. The accuracy of the scheme depends on the size of the interval under consideration. The third to fifth order accurate WENO scheme constructs an approximation to ϕ_x by a convex combination of the three different third order ENO approximations at each point. For smooth regions this results in a fifth order accurate approximation. In the vicinity of discontinuities the smoothest third order approximation is used depending on the characteristics of ϕ. Implementation details and more information is given in the books on LSM [65, 75] or the original papers cited above.

4.3.2 Temporal discretization

To evolve the fundamental level-set PDEs, Equations (4.7 and 4.8), in time on a computer we need a way to modify the values on the grid over time. This process is called *time integration* and can be done in many different ways. The simplest involves a Taylor expansion of $\phi(\boldsymbol{x}, t)$ about time t for the partial time variable. We start by defining an *initial value problem* for a general PDE

$$\frac{\partial \phi}{\partial t} = L(t, \phi(t)), \quad \phi(t_0) = \phi^0, \tag{4.20}$$

That is, the partial time derivative equals an (arbitrary) function, L, that is depending on time and the level-set function itself. The goal is to find the successive values $\phi(t_1)$, $\phi(t_2)$, ..., $\phi(t_n)$ given an initial value $\phi(t_0)$. A Taylor expansion about $(t + \Delta t)$ gives a first order approximation of the time derivative

$$\frac{\partial \phi}{\partial t} \approx \frac{\phi(t + \Delta t) - \phi(t)}{\Delta t}. \tag{4.21}$$

We combine this with Equation (4.20) and single out the next time step

$$\phi(t + \Delta t) \approx \phi(t) + \Delta t L(t, \phi(t)). \tag{4.22}$$

This scheme is known as *forward* Euler time integration because the Taylor expansion is done for $x + \Delta t$. The method only depends on known (previous in time) values of ϕ and is thus called *explicit*. Substituting the unknown function $L(\phi(t), t)$ with one of the fundamental level-set equations we can find a discrete expression to solve.

Had we instead used a backwards difference method, that is (expanded about $t - \Delta t$), we would have found the backward Euler time integration

$$\phi(t + \Delta t) \approx \phi(t) + \Delta t L(t + \Delta t, \phi(t + \Delta t)). \tag{4.23}$$

This is not an explicit method since there is a dependence on future values on both sides of the equation. The relationship between this and the next time step is instead an equation system, which implies that there is a solution. Therefore, this type of method is called an *implicit* time integration scheme. It involves the solution of an equation system to find $\phi(t + \Delta t)$. This often involves more work than an explicit method, but

has, in general, a larger stability region allowing larger time-steps as we will get back to.

Both the forward and backward Euler methods have first order accuracy in time. For some applications this may be enough, especially if speed is of the essence. However, higher order methods are readily derived using the same approach as shown above for spatial derivatives. We mention some of the more popular here and refer to [65, 75] and relevant references therein for more information.

The most common time integration schemes belong to the Runge-Kutta (RK) family. The first RK scheme is identical to the forward Euler method. Several second order RK methods exists. One of the most commonly used is Heun's method

$$\begin{aligned}\phi^{(1)} &= \phi^n + \Delta t L(\phi^n)\\ \phi^{(2)} &= \phi^{(1)} + \Delta t L(\phi^{(1)})\\ \phi^{n+1} &= \phi^n + \Delta t \tfrac{1}{2}(\phi^{(1)} + \phi^{(2)}).\end{aligned} \tag{4.24}$$

A third order version is readily constructed, but the fourth order method (RK-4) is so popular that it is sometimes called *the* RK method. RK-4 is reasonably simple to implement and yields a good trade off between complexity and accuracy. However, for LSM the high order spatial differential (WENO) tends to lose its accuracy in difficult regions, dropping from fifth to third order. In this case there is little to gain from using a fourth order time integration method [65].

TVD Runge-Kutta schemes

One group of time intergrators that have proven especially useful in physics and engineering problems are the *total variation diminishing* (TVD) methods. The total variation of a quantity u is measured as

$$TV(u) = \int \left|\frac{\partial u}{\partial x}\right| dx, \quad \text{or in the discrete case} \quad TV(u) = \sum_j \left|u_{j+1} - u_j\right|,$$

and a method is said to be TVD if it is bounded in the following sense

$$TV\left(u^{n+1}\right) \leq TV\left(u^n\right), \tag{4.25}$$

when first order forward Euler time integration is used. The boundedness ensures that no spurious oscillations can occur and the solution becomes physically realizable by construction. The TVD concept was disovered by Harten in 1982 [37]. Osher and Shu [78] then introduced a family of time integrators based on the popular Runge-Kutta methodology that retain the TVD property at higher orders. These integrators are based on the fact that a convex combination operation is TVD for non negative coefficients. Therefore, the convex combination of consecutive Euler steps is TVD by construction.

Heun's method in Equation (4.24), is a TVD-RK method since it has positive coefficients. A third order method, TVD-RK3 [78], reads

$$\begin{aligned}\phi^{(1)} &= \phi^n + \Delta t L(\phi^n) \\ \phi^{(2)} &= \tfrac{3}{4}\phi^n + \tfrac{1}{4}\phi^{(1)} + \tfrac{1}{4}\Delta t L(\phi^{(1)}) \\ \phi^{n+1} &= \tfrac{1}{3}\phi^n + \tfrac{2}{3}\phi^{(2)} + \tfrac{2}{3}\Delta t L(\phi^{(2)}).\end{aligned} \tag{4.26}$$

A fourth order method has been proposed as well [36], but the implementation becomes quite involved, and as discussed earlier the increased accuracy might be wasteful if the spatial discretization has a lower effective order.

4.4 NUMERICAL STABILITY

We have now gathered enough information to compute the evolution of a level-set by the iterative solution of a discretized PDE. The approximations involved converge to their continuous counterparts as the step size goes to zero. However, we must also consider properties apart from accuracy. Numerical stability is a general term in discrete mathematics that describes how errors propagate through an algorithm. Although a specific time, or spatial discretization, has a given accuracy (as its step size tends to zero) it is not guaranteed that the overall computations are *convergent* [80]. This is the same as to say that the resulting error (exact solution − approximated solution) does not go to zero as the step-sizes in space and time go to zero. To ensure a convergent method we need, according to the Lax-Richtmeyer theorem [80], *stability* and *consistency*.

Consistency is, in this case, related to the local truncation error of the finite difference scheme. A numerical method is said to be consistent if the finite difference approximation of the discrete operator converges towards the continuous operator as the step size in time and space goes to zero [80]. A method that has order greater than 0 is therefore consistent. All the time integration schemes introduced above are consistent.

Stability, on the other hand, is a measure of how noise grows in the algorithm. This can be from round-off errors or initial conditions. A stable algorithm does not amplify noise. Mathematically, this means that the norm of the numerical solution is bounded by the sum of the norm of a fixed number of earlier time steps. For the TVD-RK methods above, we see that this constraint is heavily enforced (provided that the sub steps are stable).

4.4.1 Hyperbolic equations

Equations (4.7 and 4.8) are hyperbolic PDEs when the speed function and the velocity field do not depend on derivatives of order greater than one. In this case they are examples of so-called *Hamilton-Jacobi* equations [5]. These PDEs have the property that information is propagated along certain directions; the *characteristics*. Because

of this limited domain of dependence, special treatment is needed when a numerical approximation to the spatial derivative is sought.

As stated above, Equations (4.7 and 4.8) are algebraically identical, but physically they describe different phenomena and we use different numerical techniques to solve them. We start with the motion described by the advection in an externally generated vector field; Equation (4.7). A basic first order discretization in time gives the function value at the next time step as

$$\phi^{n+1} = \phi^n - \Delta t \, \phi_x^n \cdot \vec{V}^n. \tag{4.27}$$

The information needed to form an accurate spatial derivative ϕ_x can be found by looking in the *reverse direction of the vector field flow*. This is corroborated by the fact that information is propagated solely along characteristics. The direction of the information flow can, in this case, trivially be found from the vector field $\vec{V}$, and is then used to pick the correct domain of dependence

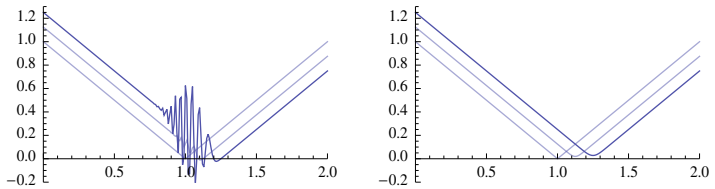
$$\phi_x^n = \begin{cases} \phi_x^+, & V^n < 0 \\ \phi_x^-, & V^n > 0 \\ 0, & V^n = 0. \end{cases} \tag{4.28}$$

Here we let $\phi_x^\pm$ denote a general derivative which favours points on the positive or negative direction along the axis in question. In practice this amounts to the inclusion of more points on either side of the current ϕ, as demonstrated earlier with forward and backward differences. This method for choosing the spatial derivative is for obvious reasons called *up-winding*. In 2D and 3D (and above) it is applied in a dimension by dimension manner. Figure 4.5 shows the hazards of not considering the correct domain of dependence and using information from "wrong" directions. In (a), the central difference scheme has a higher order truncation error than the first order one sided version used in (b), but the numerical solution is not stable, and even develops catastrophic behaviour over time.

From the introduction of consistency earlier in this section we note that the above approximations of spatial and time derivatives are consistent. But because forward Euler as well as RK-schemes introduced earlier are explicit methods their stability regions are limited. This means that to ensure that errors do not grow over time, the time step size must be restricted by the grid step size. A necessary but not sufficient requirement for stability is formulated by the CFL condition [20, 80], which states that

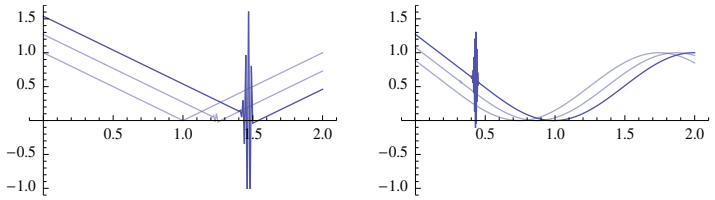
$$\frac{V \cdot \Delta t}{\Delta x} < c \tag{4.29}$$

where c is the CFL-number and depends on the specific method. For the first order Euler in Equation (4.28) the CFL-number is 1 and we get the relation $V \cdot \Delta t < \Delta x$. Numerical information is not allowed to travel more than one grid point per time step. For higher order methods this constraint can be significantly harsher and implicit methods are favourable in some cases.



(a) (b)

Figure 4.5: Advecting a sharp shape in the vector field $\vec{V}(x) = 1$ using a first order forward Euler method with a restrictive time step, $\Delta t = 0.5\Delta x$. The spatial derivatives are computed with: (a) second order central differences, and (b) first order one sided differences together with up-winding.



(a) (b)

Figure 4.6: Advection in the vector field $\vec{V}(x) = 1$ using a first order forward Euler method with an excessive time step $\Delta t > \Delta x$. (a) Catastrophic behavior quickly develops for sharp shapes, (b) but also in perfectly smooth regions.

Due to the non-linearity of Equation (4.8) explicit time integration schemes are typically used to evolve the interface in time. With forward Euler we get

$$\phi^{n+1} = \phi^n - \Delta t \, \|\nabla \phi^n\| \, F^n. \tag{4.30}$$

A conservative numerical scheme for solving PDEs was derived by Godunov in [32]. For a general Hamilton-Jacobi problem, such as the advection formulation in Equation (4.7), this scheme can be shown to produce the same results as the up-wind procedure in (4.28). Later, in [72], a formula for computing the square of the spatial derivative was derived from Godunov's method

$$\phi_x^2 = \begin{cases} \max(\max(\phi_x^-, 0)^2, \min(\phi_x^+, 0)^2), & \text{if } F > 0 \\ \max(\min(\phi_x^-, 0)^2, \max(\phi_x^+, 0)^2), & \text{if } F < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.31}$$

It is applicable whenever the direction of the gradient vector is irrelevant. When solving Equation (4.8), for example, only the norm of the gradient vector is sought and its direction is inconsequential.

Because $\|\nabla\phi\|_2 = \sqrt{\phi_x^2 + \phi_y^2 + \dots}$ this can be used to compute the norm of the gradient as needed in the PDEs. As pointed out in [54] the computation can be further simplified by considering the sign of the speed function. We let the sign function be defined as follows

$$\operatorname{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{4.32}$$

and get an expression for the squared spatial derivative that is given with even fewer evaluations

$$\phi_x^2 = \max(\operatorname{sgn}(F)\,\phi_x^-, -\operatorname{sgn}(F)\,\phi_x^+)^2. \tag{4.33}$$

4.4.2 Parabolic equations

A parabolic PDE, such as Equations (4.7 and 4.8) when the speed function or velocity field depends on derivatives of order greater than one, has no characteristics. Instead the domain of dependence is infinite. Information flows into a point in space from all other points. Moreover the speed of information is, physically speaking, infinite. This means that a perturbation at one point in space immediately affects all other points. Numerically this is accommodated by putting equal emphasis on all directions when evaluating the spatial derivatives. For example consider Equation (4.8) used for geometric smoothing $\partial\phi/\partial t = -\alpha\kappa\,\|\nabla\phi\|$. We can approximate the all-directions principle by using a central difference scheme, $\phi_x = \phi_x^0$. Here, we let the expression ϕ_x^0 denote a general central difference of which Equation (4.19) is one example.

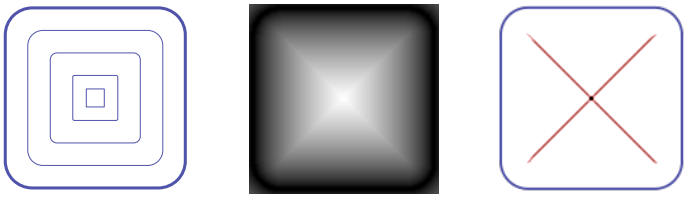
4.4.3 Vanishing viscosity solutions

Consider the following one dimensional version of the eikonal equation

$$\begin{cases} \|\nabla u(x)\| = 1, & x \in (-1, 1) \\ u(-1) = u(1) = 1. \end{cases} \tag{4.34}$$

Is there a unique solution u to this problem? It is easy to see that there is no continuously differentiable solution that fulfills both constraints in a classical sense. At some point there must be one, or more, discontinuities in the function's derivative.

One possibility would be to only consider smooth functions, but in the case of the dynamic level-set formalism, we note that even if the implicit function ϕ is initially differentiable and describes a smooth interface this may change over time. As the interface evolves it is possible, and likely, that sharp features will emerge and make ϕ non-differentiable over the interface. An example of this is shown in Figure 4.7(a). Also note that away from a smooth interface the level-set function may contain discontinuities. In fact, any distance function describing a closed curve contains at least one point for which the function's gradient is discontinuous. Examples hereof are



(a) (b) (c)

Figure 4.7: (a) An initial smooth interface (thick) can still develop sharp corners (discontinuities) as the interface evolves; in this case inwards in the normal direction. (b) The distance field of the initial shape. (c) The distance field of any closed curve contains discontinuous gradients by construction, indicated with red.

the center point of the circle, the tip of the cone, and in general points which lie on a medial axis, see Figure 4.7(b) and (c).

To circumvent the problem with differentiability we introduce a tentative solution

$$u(x) = \begin{cases} -x, & x \in [-1, 0] \\ x, & x \in [0, 1]. \end{cases} \tag{4.35}$$

which fulfills Equation (4.34) almost everywhere: this is called a *weak solution*. The function is depicted in Figure 4.8(a). The term weak solution describes a generalized solution to an ODE or a PDE where the constraint on differentiability is weakened by a transformation of the problem into a derivative free form[26]. A weak solution has the desirable property that for smooth regions it equals the "real" solution. On the downside a weak solution is not unique so additional constraints are needed in order to single out a distinct solution. This leads us to the concept of the *viscosity solution* as introduced in [21]. An brief outline of the method follows, for more information consult [5, 26], the textbooks on level-set methods [65, 75], or the original papers [21, 22]. We first recast the eikonal equation above as a time dependent initial value problem and insert a smoothing term

$$\frac{\partial u}{\partial t} = 1 - \|\nabla u(x)\| + \underbrace{\epsilon \Delta \phi}_{\text{smoothing}} . \tag{4.36}$$

This artificial viscosity term effectively regularizes the eikonal equation and thus produces smooth solutions. Now the reasoning about the viscosity solution is that we let the artificial viscosity tend to zero, $\epsilon \to 0$, and hence pick out a unique weak solution. This solution is shown in Figure 4.8(b).

It is common in fluid dynamics to add the smoothing term explicitly to better handle the presence of shocks and discontinuities. For the numerical schemes used with

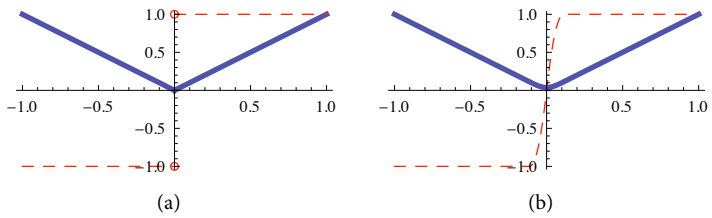


Figure 4.8: The function (blue) in Equation (4.34) and its derivative (dashed red) illustrating the vanishing viscosity solution. (a) The sharp version has a jump in its derivatives and is not differentiable at 0. (b) A regularized version of the same function has a continuous derivative and is differentiable everywhere.

LSM this is normally not done as there is already a large degree of numerical dissipation built in. Adding extra smoothing would typically cause excessive dissipation.

To better understand the dissipative effect we can study the numerical schemes from a different point of view. For example, instead of seeing the first order forward Euler as producing a solution with an error term that is $O(\Delta t)$ it is possible to see this as an *exact* solution, but to a slightly different PDE. For a first order one-dimensional Euler scheme this other PDE contains a term which implicitly functions as viscosity, see for example [4].

We finally mention that the dissipative effects come from the even derivatives in the truncation error of the discrete differential operator. The odd derivatives introduces a different numerical effect that shows up as "wiggles" in front and behind waves. This is called *dispersion*. Together dissipation and dispersion are called *numerical diffusion* and we note that this effect is less pronounced for higher order schemes. Figure 4.5 shows, albeit in a different setting, dissipative and dispersive behaviour, respectively.

Finding the viscosity solution

Although interesting, this theory is of little use if it is difficult to implement on a computer or renders computations prohibitively slow. Fortunately, the upwind method in section 4.4.1 as well as Godunov's method are constructed to pick out the physically plausible vanishing viscosity solution. More information on level-set methods, PDEs and viscosity solutions is available, in varying detail, in [4, 26, 65, 75].

4.5 MAINTAINING A SIGNED DISTANCE FUNCTION

The level-set function ϕ is best kept as a signed distance function (Equation 4.10). Unfortunately the iterative solution to the initial value problem in Equation (4.20) does not ensure this. Thus the discrete distance field must regularly be *re-initialized*

to guarantee that Equation (4.12) is fulfilled. The first attempts to do so [82], used the initial value formulation of the eikonal equation [72]

$$\frac{\partial \phi}{\partial t} = \operatorname{sgn}\left(\phi^0\right) \cdot (1 - \|\nabla \phi\|), \tag{4.37}$$

which they solved to steady state. Here, the sign function effectively chooses the correct solution for both Ω^+ and Ω^- by looking at the values of ϕ (before they change). Numerical tests have shown [68] that the discretization of the sign function benefits from smoothing such that $\partial\phi/\partial t = S(\phi) \cdot (1 - \|\nabla\phi\|)$, with

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + \|\nabla\phi\|^2 \ (\Delta x)^2}}. \tag{4.38}$$

The re-initialization of ϕ is done by iteratively solving the initial value problem until a steady state is reached. From the method of characteristics we have that the correct updates are given in an increasing (per iteration) radius out from Γ. The CFL condition furthermore tells us that the solution moves at most $c\Delta x$ per iteration. Thus we can guarantee a correct solution over an $m \times m$ grid after $\sqrt{2} \cdot m/c$ iterations, making the method scale quadratically in the number of grid points, $O(n^2)$, since $n = m \times m$. The major advantage of the initial value approach is the ability to apply higher order differentials (*e.g.* ENO/WENO), see [65] for implementation details.

The eikonal equation can also be solved as a boundary value problem, and this has been done with great success through a number of papers. Tsitsiklis [84] and Sethian [74] independently came up with the same method to solve the eikonal equation using only n operations for n grid points. At each step, the optimal node must be updated for this complexity to hold. Both methods use sorting to quickly find the correct node making the overall complexity $O(n \log n)$ in time. Sethian called his algorithm the *fast marching method* (FMM), which is the name we will use. If the sorting of the update in FMM is omitted, an asymptotically sub-optimal method is given. In [44] such an algorithm is proposed and shown to be very fast in practice even if its complexity cannot be bounded.

Several versions of *sweeping* algorithms have also been proposed that are significantly faster in some cases, especially for simpler solution fronts. In [23], an early method was proposed, but in the level-set community, a more recent approach [88] is often used. These methods are fast and particularly simple to implement. Additionally, all sweeping methods can easily be parallelized, which has not yet been done for FMM due to the sorting involved. But, due to their nature, the sweeping methods are best suited for regular domains.

In paper [64] we propose a method that accurately computes a distance function. Our approach is not derived from the viscosity solution to the eikonal equation (4.12). Instead, our work is based on [35, 71] that use an integral formulation of distance. We further discuss distance functions in Chapter 6, and for now conclude that our method is readily used to re-initialize a distance field.

4.6 BRINGING IT ALL TOGETHER

We end this chapter with a simple example showing the method in use. But first, we introduce some notation. When discussing general storage concepts we do not fix dimension unless necessary, as level-set methods are not limited to specific dimensions. We use a loose C++ style for the pseudo code listings. With this in mind a minimal example of a level-set data structure is given.

```
1 class LevelSet {
2     Grid<float> phi; // The sampled implicit function
3     float h; // The grid step size
4     float t; // The current time
5     Propagate(Grid<float> F, float dt); // Eqns. (4.8) + (4.20)
6     Advect(Grid<Vector> V, float dt); // Eqns. (4.7) + (4.20)
7     Reinitialize(); // Equation (4.12)
8 };
```

Listing 1: A level-set data structure

The statement *Grid<float> phi;* on line 2 in the listing above declares the variable *phi* to be of the class *Grid*. The type names are meant to be self explanatory. The grid, in this case, gives the possibility to retrieve data at indexed positions, *i.e.* in two dimensions $phi(i, j)$ gives the value stored at the coordinate (i, j). Furthermore, as indicated by the ending $< \cdot >$, the Grid is thought of as a multipurpose type which can hold different types of data[2]. In this case, phi holds floating point values that encode the distance function.

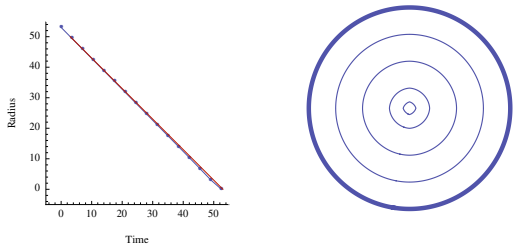
The grid step size h is used when evaluating finite difference approximations to the spatial derivatives in the level-set equations. In a similar way the variable t is keeping track of the current time. Because the motion is an initial value problem the corresponding speed function F and vector field V are also time dependent and t is used to find the correct instance in time.

The *Propagate* function solves the fundamental level-set equation (4.8) in the initial value setting (4.20) such that the current time is advanced to $t + dt$. If the requested time step is larger than permitted by time step restrictions such as dictated by the CFL condition the propagate method is responsible for internally dividing it into smaller sub steps. The *Advect* function similarly moves the interface but uses the vector form in Equation (4.7). Finally, the function *Reinitialize* resets the distance grid *phi* so that it obeys Equation (4.12).

To use the level-set methodology we can now employ the following cycle

- ⇝ Solve Equation (4.20) over the sampled domain for one of the level-set equations (4.7 and 4.8); function *Propagate* or *Advect*
- ⇜ Re-initialize the distance values, function *Reinitialize*

2. As in the C++ concept of *templates*



(a) (b)

Figure 4.9: Shrinking a circle with unit speed function with first order approximations. (a) The radius as a function of time. Blue – simulated, red – exact solution. (b) The curve itself at successive time steps.

4.6.1 Eroding a circle in the normal direction

In Figure 4.9 we apply the level-set simulation cycle to a simple example using the first order approximations introduced in this chapter. The figure shows the result of shrinking a circle with unit speed function, $F = 1$. Because of the, in this case, trivial analytic solution we can compare the simulation to exact data in (a), the graph shows the radius of the circle over time. Some of the corresponding level-sets are shown in (b), the thick curve indicate the initial condition.

As is expected, the simulated radius deviates from the exact solution. This is an effect of the approximations introduced in both space (first order up-winding), and time (forward Euler). The linear estimation of the differential operators also manifests itself clearly in the last (innermost) contour in Figure 4.9(b). The correct solution should be a perfect circle, but the simulated version is starting to look more like a diamond.

4.7 SUMMARY

In this chapter the level-set theory was introduced. We showed first how to sample the geometry in space and found that a signed distance function was a good choice. We then continued to describe how drive these dynamic surfaces, and arrived at the fundamental level-set equations. These continuous PDEs require delicate handling due to certain specific physical properties. This also carried over to the discrete operators used to numerically solve the PDEs.

Many factors affect the numerical stability of a level-set computation. In particular it was exemplified that when not adhering to the correct discretizations, catastrophic behavior can, and is likely to, develop over time.

CHAPTER 5

IMPLEMENTING LEVEL-SET METHODS

From a practical point of view the specific LSM implementation can be of tremendous importance. Running times for large simulations can differ in hours and even be impossible because of the sheer amount of data. A simulation domain that is $500 \times 500 \times 500$ grid points large requires, using the most basic time integration and single precision, two buffers of roughly 475 MB each. If the interface is advected in a velocity field, then 3×475 MB more is needed just to hold this data. Add simulation of the vector field and/or a high order time integration scheme and the number of buffers needed increase dramatically. The problem is twofold – there is both a need to store large simulation domains, and at the same time all the data must be efficiently processed.

5.1 INTRODUCTION

Part of the work in this thesis has been focused on implementing data structures with a light memory footprint without sacrificing versatility or speed. This has been a delicate line to balance. If one focuses too much on memory efficiency chances are that this comes at the price of speed, and the other way around.

This chapter is an introduction to data structures used in LSM and especially the data structures used in papers [41, 61]. We begin this chapter with the introduction of a localized level-set method and a minimal data structure that goes with it. From this starting point we develop more sophisticated grids. This is done in 2D to keep concepts clear and minimize the notation. However, all the data structures described in this chapter generalize to any dimension, but not all of them do so gracefully due to excessive memory constraints.

5.1.1 Computational aspects

When implementing LSM on a computer many factors on different levels affect the efficiency of the result. First one needs to identify a problem statement. Is the main interest to represent static geometry with the level-set method? For simulations or collision detection with rigid geometry the level-set method is ideal. The key word *static* indicates that the geometry is not deforming. In this case significant pre-processing can be justified. In an "opposite" scenario the efficiency of a *continuous* simulation is most important. Segmentation and water simulation are examples from this class.

Every situation needs different considerations. We start by listing some efficiency related factors.

- Algorithmic complexity in the number of grid points for the problem at hand

Algorithmic complexity dictates the asymptotic running times and is always the first place to look for improvements, especially for continuous problems. Special care should also be taken with respect to side effects (*e.g.* re-initialization). For static problems specific implementation details can also be of significant importance.

- Sampling and resolution topics

The question to ask is accordingly: Is it possible to use less sample points and yet obtain the desired accuracy? Sampling and resolution have inspired much work in the level-set community. It is the source of the *narrow band* methods as introduced by [2, 19, 86], and has also led to adaptive grid methods such as [49, 79].

- Access times for data retrieval from main memory to the processor.

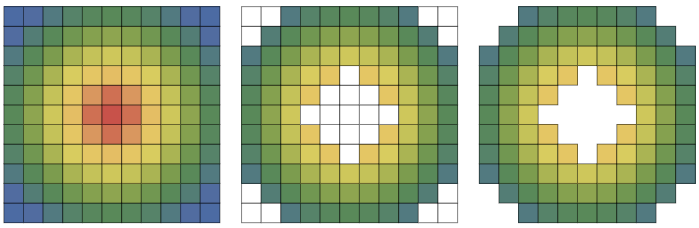
Access times and cache coherence of a data structure are crucial factors when dealing with large LSM geometry. Especially cache performance has become increasingly important in later years because of the relatively limited bandwidth between the processor and (much larger) main memory. The cache holds on to pieces of memory with the hope that the processor will need them soon again. Because it is considerably faster to access memory that resides in the cache than fetching it from main memory this is a favourable strategy. However, the size of the cache is limited (as it is very expensive), and the few slots of the cache regularly gets over-written with newer data. Thus, the boost that the cache can deliver requires the processor to use the same piece of data multiple times over a sufficiently short time span. In short, locality of computations (with respect to positions in main memory) increases cache performance.

- Is the problem parallelizable?

The current development in computer hardware suggest that computers will become more and more parallel. This is partly because the performance of traditional CPUs have reached a level where significant amounts of energy are needed to increase performance. On the other hand, low cost but high-performance accelerator units (GPUs) are readily available. A good data structure should be designed with this in mind.

5.2 A LOCALIZED LEVEL-SET METHOD

We begin this hands-on chapter with the introduction of one of the major efficiency improvements to the level-set method as seen from a practical point of view: a localized level-set using the *narrow band method* [2, 19]. The idea is based on the fact that the information given over the embedding, Ω, for some purposes is more than actually needed. In many applications the interest lies only in the position of the interface, Γ. The shape of Ω is arbitrary as long as it is possible to recover Γ.



(a) (b) (c)

Figure 5.1: A "dense" grid with sample values (a). To reduce the computational cost a mask (b) over the dense grid can be used to indicate "active" grid points. (c) To reduce both time and space complexity a fully sparse grid is needed.

With LSM, the geometry is represented by a function sampled on a grid. We prefer ϕ to be a signed distance function for different reasons as discussed in Chapter 4. Since LSM is an implicit representation ϕ will be defined over the full domain. For a curve this is normally a finite rectangle-shaped part of $\mathbb{R}^2$, represented by a (dense) grid. An example is shown in Figure 5.1(a).

From an efficiency point of view the information far away from Γ is often a *burden* because for all these grid points the implementation needs to 1) provide storage, and sometimes also 2) perform computations according to some numerical scheme. The fact that the original LSM approach [66] scaled with the size of the embedding domain made early implementations memory consuming and slow. This is a reputation that, somewhat undeservedly, has stuck with the method.

One solution to the scaling-problem is to discard all sample points that are not needed. For a signed distance function this can be done by considering the magnitude (distance) of the sample points, and only keep those that are closer than some constant. See Figure 5.1(b) for a depiction where active grid points in a dense grid are chosen based on their distance to the interface. Sufficiently many sample points around the interface need to be kept in order to be able to perform an accurate reconstruction of Γ. When using finite differences to compute approximations to the differential operators this means that the computational domain must be wide enough to support the stencil of the approximation, see Figure 4.4.

5.3 THE NARROW BAND METHOD OF PENG ET. AL

The narrow band approach was, to the best of our knowledge, first described in [2, 19] and is further refined in many consecutive papers some of which are included in this thesis [41, 61].

In its simplest form the narrow band method performs its computations only on the grid points for which the magnitude of ϕ is smaller than a constant δ. We write

this band as the set in n-dimensional space $\{\boldsymbol{x} \mid \text{abs}(\phi(\boldsymbol{x})) < \delta\}$, or discretely in 2D as the grid points

$$\{(i, j) \mid \text{abs}(\phi_{i,j}) < \delta\}. \tag{5.1}$$

Different ways of storing these narrow band grid points exist; a few examples of this are illustrated in Figure 5.1. In the first papers [2, 19] a dense grid was utilized together with a mask on the form of equation (5.1). This means that the storage complexity is $O(m^2)$ where m is the grid length size. The computational complexity though, drops to $O(S)$, where S is the number of grid points in the narrow band. For δ significantly smaller than m this means that computations now scale with the size of the interface instead of its embedding. This can give a major speed-up, but care must be taken when maintaining the mask data structure. The rebuild of the mask will, if implemented naively have $O(m^2)$ time complexity. In Figure 5.1(b) a dense grid with a subset of "active" grid points is illustrated.

In [68] Peng *et al.* introduced a narrow band scheme intended for level-set simulations that has since become widespread. Their method divides the computational domain into several bands. Then, based on this classification a clamped time integration scheme is applied that reduces numerical oscillations at the boundaries of the truncated embedding. First, let a clamped version of the initial value problem from the last chapter (Equation 4.20) be

$$\frac{\partial \phi}{\partial t} = c(\phi) \cdot L(t, \phi(t)), \quad \text{with } \phi(t_0) = \phi^0, \text{ and } c(\phi) \in [0, 1]. \tag{5.2}$$

If $c(\phi) = 0$ then ϕ does not change over time (no motion) and if $c(\phi) = 1$ then Equation (5.2) becomes identical to Equation (4.20). Peng *et al.* suggested the following smooth transition between 0 and 1

$$c(\phi) = \begin{cases} 1 & \text{if } |\phi| \leq \beta \\ \frac{(|\phi| - \gamma)^2 (2|\phi| - 3\beta + \gamma)}{(\gamma - \beta)^3} & \text{if } \beta < |\phi| \leq \gamma \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

This function is plotted in Figure 5.2. It makes use of the constants (β, γ, δ) to define the three concentric computational bands:

$$\begin{cases} \beta\text{-band} & \text{for } |\phi| \leq \beta \\ \gamma\text{-band} & \text{for } |\phi| \leq \gamma \\ \text{entire-, expansion- or } \delta\text{-band} & \text{for } |\phi| \leq \delta. \end{cases} \tag{5.4}$$

The purpose of the β- and γ-bands or tubes is to efficiently provide the domain in which the time integration in Equation (5.2) is carried out. The δ-band, next, allocates enough space around the γ-tube so that the computational domain can move during advection. In Figure 5.3(a) we show the different tubes overlaid on a continuous function, and in (b) the discrete counterpart. According to Peng *et al.* the values inside the β- and γ-tube are signed distances, and values outside this band are clamped to

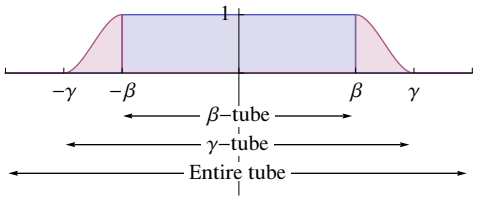
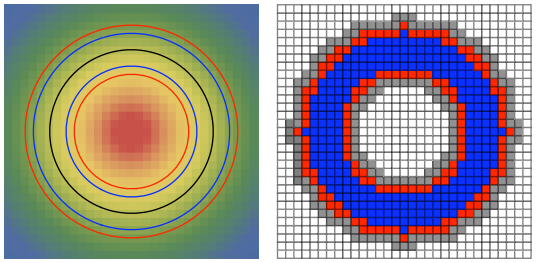


Figure 5.2: A cut-off function around the computational tube can reduce oscillations at the boundaries. The width of the entire tube is usually 1 grid point (Δx) larger than the γ-tube to facilitate the inclusion of new grid points as the interface moves.



(a) (b)

Figure 5.3: The different tubes around the interface. (a) Shows the continuous tubes: red – γ-tube, blue – β-tube, and black – interface. (b) The discrete tubes: *i.e.* (i, j) such that $\phi_{i,j} < c_i$ where c_i is one of β, γ. The gray coordinates indicate that the entire tube is wider than the γ-tube so that the interface has room to move during advection.

$\pm\gamma$. Even though this means that some of the special properties of the signed distance function are lost, the clamping ensures a consistent approximation of ϕ outside of the computational band.

5.3.1 *A simulation step which is linear in time*

With the clamped time integration in Equation (5.2) only values inside the narrow band are updated during simulation. A naive implementation loops over the full grid and checks the magnitude of the level-set function to see which grid points that are active, and updates only those. This is faster than updating all grid points, but still asymptotically $O(n^2)$; the method scales with the size of the grid instead of the interface. In order for a simulation step to be efficient, extra data structures are required [68]. First keep a *coordinate list* holding the grid points included in any of the bands. Additionally, a *mask* on the same size as the underlying grid is used to label all grid points according to what band they belong to, or inside/outside. A simulation step now

loops over the coordinate list and only updates grid points inside the computational band. This method scales with the size of the interface.

The narrow band method of Peng *et al.* has the following solving cycle

- ⇝ Solve Equation (5.2) inside the γ-tube for one of the fundamental level-set equations
- ↓ Re-initialize the distance values over the entire narrow band
- ⇜ Rebuild the narrow band by updating the coordinate list and mask.

In order to support the new functionality some additions to the initial data structure in Listing 1 are needed. A minimal narrow band data structure is

```
1 class NarrowBandLevelSet {
2     Grid<float> φ // The sampled implicit function
3     Grid<Type> M  // The mask
4     List<Coord> C  // The coordinates in the band
      ⋮
5     void Rebuild()
6 };
```

Listing 2: A narrow band level set data structure

The dots (⋮) implies that the relevant functionality in Listing 1 is preserved. Additionally, the new "rebuild"-method is responsible for moving the computational domain when needed.

The level-set equations can now be solved inside the computational band efficiently by looping over the coordinate list. This gives rise to a level-set simulation method that is linear in time (with respect to the size of the interface). The rebuild-method in the original paper [68], however, scales with the size of the full grid, *i.e.* it has complexity $O(m^2)$ for a two dimensional grid. This term becomes dominant for large grids and can severely hamper performance, see for example [59]. A method which is linear in the number grid points in the narrow band is preferable. In paper [62] we proposed one such method which is described below.

Rebuilding the narrow band data structure

After each step of solving Equation (5.2) and subsequent re-initialization, the narrow band data structure must be rebuilt. This is due to the fact that the interface and hence the tubes move during advection. The gray grid points in Figure 5.3(b) show the extra space allocated around the γ-tube that enables the band to move. A width of a single grid point around the band is reasonable given the constraints on explicit time integration methods. [1]

In Listing 3 a fully linear rebuild algorithm is outlined; it is taken from [62] but slightly revised. The algorithm works by first tagging all grid points inside the com-

1. Remember that the CFL-condition restricts explicit time integration to move less than one grid point per time step.

putational tube, that is all the grid points in the coordinate list, according to their signed distance value. Then all neighbors are inspected: if necessary, they are added to the expansion band, else, they are ignored. *All* neighbors are inspected and not just neighbors to values in the gamma band. This is to ensure a correct expansion band. If the gamma band is wide enough ($\gamma > \beta + \sqrt{2}\Delta x$) and the level-set function is reinitialized to exact values then the expansion band classification (line number 11-17) can safely be moved to the end of the else statement on line number 8. This improves performance slightly. But, since the reinitialization often is only approximately solved the safe guard is sensible for thin tubes. Furthermore the penalty for doing so retains the same linear asymptotic complexity.

```
   Input: Coordinate list C, mask grid M, and distance grid φ
   Output: Coordinate list C′, modified mask M and clamped distances φ
 1 foreach (i, j) ∈ C do
 2     if |φ_{i,j}| < γ then
           /* classify the computational bands                          */
 3         if |φ_{i,j}| < β then
 4             M_{i,j} ← β
 5         else
 6             M_{i,j} ← γ
 7         C′ ← (i, j)
 8     else
 9         M_{i,j} ← outside
10         φ_{i,j} ← sgn(φ_{i,j}) · γ
       /* classify the expansion band                                   */
11     foreach (p, q) ∈ Neighbor(i, j) do
12         if |φ_{p,q}| < γ and M_{i,j} = outside then
13             M_{i,j} ← δ
14             C′ ← (i, j)
15         if |φ_{i,j}| < γ and M_{p,q} = outside then
               /* This is the only way to enter the band                */
16             M_{p,q} ← δ
17             C′ ← (p, q)
```

Listing 3: A linear rebuild of the narrow band data structure.

Figure 5.4 shows a complexity plot indicating the behaviour of the method in [68] compared to ours. We also compare to the more complex rebuild method in [58, 59], applied to the narrow band data structure in Listing 2.

5.3.2 *Dilation*

To more succinctly describe the rebuild procedure we split it into two distinct parts. First, a classification/pruning step is performed, which updates the mask values and

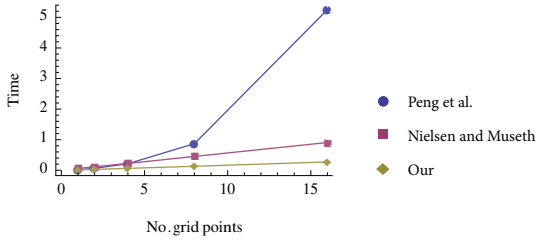


Figure 5.4: Rebuild complexity for successively larger grids. The plot shows the time to rebuild the data structure vs the total number of grid points in the narrow band.

removes coordinates that should no longer be kept in the list. Then, comes a *dilation* step, which expands the width of the computational band to allow for interface movement. In Listing 3 the classification/pruning corresponds to lines 2-10 and the dilation to lines 11-17.

In [59] a dilation algorithm for fully sparse grid structures was proposed. The method can also be applied to a narrow band grid such as the one in Listing 2. We describe it here to differentiate between to our version in Listing 3.

The method performs the dilation in a column by column manner. In two dimensions it first dilates in the x-direction (between columns), and then in the y-direction (along each column) . Let a column be denoted $c_i = \{j_1, j_2, \dots\}$ where i is the column's x-index and $\{j_1, j_2, \dots\}$ is the set of sorted y-indices of the column. Also, denote the sorted set of all columns in the grid with C. A conservative dilation of width $h\Delta x$ can then be performed as follows.

First, let column c'_i be the union of c_i with its h preceding and succeeding columns: $c'_i = \bigcup_{k=i-h}^{i+h} c_k$. The inter-column dilation is then contained in the set $C' = \{c'_k\}$, for $k = i_{min} - h, \dots, i_{max} + h$. Here i_{min} and i_{max} are the indices of the first and last columns of C respectively. For columns c'_i that have no corresponding column $c_i \in C$, that is c'_i with $i < i_{min}$, or $i > i_{max}$, the column indices can still be determined, and the set of y-indices is the empty set. Finally, dilate each new column, c'_i, in the y-direction by h grid points ($c''_i = c'_i \oplus h$), the final dilation is then given as $C'' = \{c''_i\}$. In Listing 4 we show the pseudo code for a one grid point dilation of a 2D grid using this method. Figure 5.4 shows how this dilation method compares to the one in [62] (Listing 3) and the original method proposed by Peng *et al.* [68].

```
1 foreach c_i ∈ {c_{min-1}, C, c_{max+1}} do
2     c'_i ← (c_{i-1} ∪ c_i ∪ c_{i+1})  // dilate between columns
3 return C'' ← {c'_i ⊕ 1}  // dilate along columns
```

Listing 4: Dilation by one grid point.

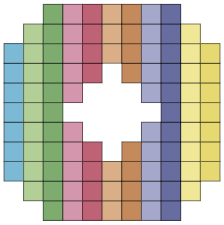


Figure 5.5: Grid points color coded by x-index value to indicate redundancy in grid storage.

5.4 FULLY SPARSE DATA STRUCTURES

Inspired by the sparse data structures used in linear algebra [25] we turn our attention to possible improvements to the data structure used in the narrow band method of Peng *et al.*, see Listing 2. Many of the popular sparse matrix data structures are possible candidates when storing sampled level-set data. We will consider coordinate storage, compressed column storage and blocked storage. Each of these storage types have their positive sides and drawbacks.

5.4.1 *The coordinate grid*

A natural starting point is the fact that the mask in Listing 2 is actually only *interpreting* the values in the grid and as such the mask is redundant. The narrow band concept only needs the coordinate and a value for each grid point in the band. Figure 5.1(c) shows such a grid. We trivially make the transition to a fully sparse grid by considering the following pseudo code

```
1 class CoordinateLevelSet {
2       List<Coordinates> C
3       List<Values> φ
        ⋮
4 };
```

Listing 5: A data structure storing coordinates coupled to values

This type of storage is called the *coordinate grid* because it explicitly stores coordinates and values. Sometimes the need arises to store auxiliary, or *subordinate*, values at each grid point; for example uv-coordinates, mass, or velocity. With a coordinate grid any number of subordinate value lists can be used together with the coordinates of the grid.

In two dimensions the coordinate grid is equivalent to the coordinate matrix storage as described in [25]. To exemplify we show how to store the following sparse matrix

$$\boldsymbol{M} = \begin{pmatrix} 5 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 8 \\ 7 & 0 & 0 & 3 \end{pmatrix}. \tag{5.5}$$

As a coordinate matrix the data can be represented by two lists:

$$\text{values: } \{5, 7, 2, 1, 2, 8, 3\} \tag{5.6a}$$

$$\text{coordinates: } \{(0,0), (3,0), (0,1), (1,1), (2,2), (2,3), (3,3)\} \tag{5.6b}$$

if we use the convention that row indices increase downwards. In this example we have sorted the coordinates such that values in the same column are consecutive. In principle this is not required, but many operations, searches, neighborhood queries, *etc.*, are simplified or can be performed more efficiently when the data is kept sorted.

In the coordinate grid each grid point value is explicitly connected to a coordinate. This is beneficial when the grid moves, as there is no "box" limiting the range of motion. The simulations can be truly out-of-the-box as demonstrated in [41, 58]. Any grid structure that explicitly stores coordinates shares this feature.

5.4.2 *The compressed sparse column storage*

The compressed row matrix storage (CSC), see [25], stores data along columns of a matrix and compresses the indices in the process. In a level-set grid (or any matrix) the values of each column share the same x-index. Figure 5.5(a) illustrates this redundancy.

In the CSC-format the matrix in Equation (5.5) is represented by three lists:

$$\text{values: } \{5, 7, 2, 1, 2, 8, 3\} \tag{5.7a}$$

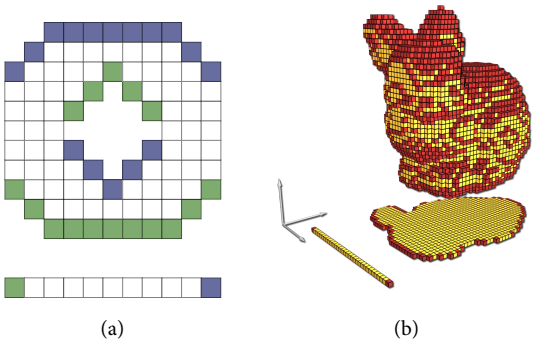
$$\text{row indices: } \{0, 3, 0, 1, 2, 2, 3\} \tag{5.7b}$$

$$\text{column pointers: } \{0, 2, 4, 5\} \tag{5.7c}$$

As for the coordinate storage, the value array holds all the non-zero elements of the matrix. The row indices of the non-zero values are also explicitly stored. Then, the column pointer list encodes two things. It explicitly encodes pointers into the other two arrays for the start- and end-positions of each column. Additionally, with its size m, it implicitly identifies each column-index and numbers it from $[0, \ldots, m-1]$. In this way the row and column indices are coupled to produce a coordinate together with a value.

5.5 HIERARCHICAL COMPRESSION TECHNIQUES

In the specific case of narrow band level-set grids there is actually more redundancy in the data than the CSC format can handle. Figure 5.6 shows how values along each



(a) (b)

Figure 5.6: Hierarchical compression of indices. (a) A two dimensional grid first compresses indices along columns, then along rows. (b) A three dimensional grid also compresses indices along the third dimension.

column not only share x-index, because of the form of the narrow band, they also have *consecutive y-indices*. This property applies for grid points *inside the narrow band* along *any dimension* and has been exploited through the recursive application of hierarchical index-compression [41, 58].

Figure 5.6(a) shows the strategy in two dimensions. The column-indices can be compressed by noting the start- and end-points, and then, the very same compression can be applied for the (single) resulting row as well. In (b) a three dimensional grid is compressed, in this case the recursive compression starts along the third axis.

We now continue to describe two different grid storage formats that use hierarchical compression. Although different in practice, in this text we focus on their similarities in order to increase readability.

5.5.1 *The DT-grid and H-RLE data structures*

The dynamic tubular grid (DT-grid) is one data structure that recursively compresses indices hierarchically [58, 59]. We briefly describe the 2D version of the data structure here. For more information see the original paper.

In a DT-grid segments inside the narrow band that have consecutive indices are called *connected components*; these are encoded with start and end values, see Figure 5.6(a). The connected components along a specific dimensions define a *projection column*. An n-dimensional DT-grid is formed by the recursive application of projections. In two dimensions this amounts to a set of projection columns formed along the y-axis, and a single projection row along the x-axis.

The DT-grid for our test matrix (5.5) is

$$\text{values: } \{5,7,2,1,2,8,3\} \tag{5.8a}$$
$$\text{projection col. in x: } \{[0,4]\} \tag{5.8b}$$
$$\text{projection cols. in y: } \{\{[0,0],[3,3]\},\{[1,2]\},\{[2,2]\},\{[2,3]\}\} \tag{5.8c}$$

A projection column, indicated by the double curly braces, is a list of lists. The actual memory storage must, in addition to the components themselves, also store the number of connected component inside each projection column.

The Hierarchical run-length encoded grid (H-RLE) was proposed in [41] as a versatile design for level-sets when used in computer graphics. It bears resemblance to the DT-grid data structure above, but it also has some quite different characteristics, as discussed in [41]. The H-RLE is based on the compression of consecutive index segments, called *runs*. A run is composed of a start index and a count value encoding the number of consecutive grid points along the segment.[2] Let a run be encoded by a start index x and a count number n giving a set of indices with start position x and in total n consecutive elements, *i.e.* $[x, x+n) = \{x, x+1, \ldots, x+(n-1)\}$.

A run in itself is dimensionless, the run $[1,4)$ stores the indices $\{1,2,3\}$, but does not reveal their dimension. In an H-RLE-grid the run-length encoding is applied recursively to all dimensions in order to achieve the most efficient compression, see Figure 5.6. Thus, in 2D, each column is compressed as a set of runs, and their reduction on the X-axis is stored as the resulting single set of runs.

In two dimensions the H-RLE grid for the example matrix in Equation (5.5) is stored as a list of values together with two lists of runs:

$$\text{values: } \{5,7,2,1,2,8,3\} \tag{5.9a}$$
$$\text{x-runs: } \{[0,5)\} \tag{5.9b}$$
$$\text{y-runs: } \{\{[0,1),[3,4)\},\{[1,3)\},\{[2,3)\},\{[2,4)\}\} \tag{5.9c}$$

In this presentation the difference between these two grids is very small, and in the implementations used for this chapter their performance is similar. This is also supported by the small test case in Figure 5.7. When it comes to random access both H-RLE and the DT-grid provide additional acceleration structures that allow for logarithmic access.

5.6 BLOCKED STORAGE

In sparse linear algebra blocked matrices are well known. In the level-set community their use was first suggested in [10] but no actual results were shown. Recently, a blocked storage for LSM, dubbed the DB-grid, has successfully been used in visual effects production [52, 55, 56] but no benchmarks are yet reported.

2. We here only consider the compression of indices which is a slightly simplified view.

Using the example matrix from Equation 5.5 and re-writing $\boldsymbol{M}$ in 2-by-2 blocks with zero blocks denoted with $*$ we get:

$$\boldsymbol{M}_b = \begin{pmatrix} \begin{pmatrix} 5 & 2 \\ 0 & 1 \end{pmatrix} & * \\ \begin{pmatrix} 0 & 0 \\ 7 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 8 \\ 0 & 3 \end{pmatrix} \end{pmatrix}. \tag{5.10}$$

We can now treat the *blocks* as a coordinate grid, an H-RLE-grid, or any of the other grid types above. To exemplify we store the block coordinates together with the values in the non-zero blocks as a blocked coordinate storage matrix

$$\text{values: } \{5, 0, 2, 1, 0, 7, 0, 0, 2, 0, 8, 3\} \tag{5.11a}$$

$$\text{coordinates: } \{(0,0), (0,1), (1,1)\} \tag{5.11b}$$

Here we let the sub-blocks be implicitly defined as every four entries in the value array. The block storage explicitly stores a relatively large number of superfluous zeros for this test matrix. This is called the *fill-in* of the matrix. For a fixed block size and large matrices, however, this number becomes small when compared to the total number of blocks.

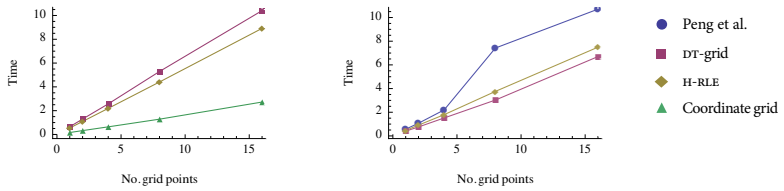
Block storage has many favourable traits. Indices are efficiently compressed by the implicit storage order inside each block. The scheme is relatively simple to implement because of its regularity. Additionally, random access can be readily accelerated. For our test matrix, a single matrix with size 2×2 (full matrix side length divided by block size) holding pointers into the block storage provides constant time random access to the data.

However, if such an acceleration structure is used on top of the blocks the fully sparse traits are lost and the memory no longer scales linearly with the size of the interface.

5.7 EFFICIENT USAGE OF SPARSE DATA STRUCTURE

Above, we described the storage schemes for some sparse data structures. To use these data structures in level-set simulations more aspects need to be considered. For example, during a simulation step all grid points in the band need to be visited and updated. The best way of doing so adheres to the underlying 1D storage in memory and sequentially visits all grid points. Given a constant time memory access, $O(1)$, the optimal complexity for doing so is linear, $n \cdot O(1) = O(n)$. Generally, we cannot expect fast random access to points in a fully sparse grid. Therefore, more elaborate designs are needed.

Additionally, the sparse grid needs to move to keep up with the shifting simulation domain. This must be done carefully in order not to have a negative impact on the overall efficiency.



(a) Iteration (b) Dilation

Figure 5.7: Relative plots of time versus the total number of grid points for successively larger grids (a) Sequentially visiting all grid points. (b) Performing 10 steps of dilation by 1 grid point each.

Fast grid point access in simulations

An *iterator* encodes a position in the grid, *i.e.* its indices and its value, and provides methods that move the iterator over the grid. A sequential forward iterator, limits movement and can only *increment* the position one grid point forward at a time. If the data structure supports this construct well, visiting and updating the grid can be done efficiently without relying on fast random access.

For all the grids above an iterator can be constructed which has a constant time increment operation, $O(1)$. When visiting all the n grid points with such an iterator the total cost becomes linear, $n \cdot O(1) = O(n)$. Still, the cost for iterating over a more complex grid is considerable. Figure 5.7(a) show some plots where the time for iteration over successively larger grids is related to the total number of grid points in the band. In this case the trivial iteration over a coordinate grid is significantly faster than both the DT-grid and H-RLE, at the cost of a larger memory footprint.

In addition to sequential iteration, access to grid point *neighbors* is ubiquitous in simulation. See for example the computational stencils of the differential operators in Figure 4.4. If random access is slow, this will have large impact on the overall times and even computational time complexity. To efficiently access grid point neighbors during sequential access a *synchronized stencil of iterators* can be used. If the synchronization operation for the dependent iterators with respect to the base iterator is constant then linear time sequential stencil access can be achieved. This is a delicate subject; more information is available in [41, 58, 59].

5.7.1 *Maintaining a sparse data structure*

Two important features of the dense grid storage in the original narrow band data structure are 1) the ability to query points outside of the narrow band, and 2) the ability to perform fast, constant time, random access. With a fully sparse grid this is no longer possible. The existence of the mask additionally simplified the rebuild process. A sparse data structure is more difficult to maintain, and especially the rebuild procedure (Listing 3) becomes complicated.

Even though the grid points outside of the band has been discarded some of the "lost" information can be recovered. When moving the tubes and inserting new points the magnitude of a point is known to be $\pm\gamma$ since the level-set values are clamped. Also, the sign can be deduced from the neighbors, $\text{sgn}(\phi_{p,q}) = \text{sgn}(\phi_{i,j})$. This is correct for all grid points sufficiently far from the interface.

It is inherently more difficult to perform the dilation step for fully sparse grids because the process needs access to grid points *outside of the band*. The method in Listing 4 was proposed with this in mind. For a hierarchically compressed grid it is more efficient because it can consider the dilation to take place on the level of segments only. Additionally, the dilation is applied recursively, that is first, the set of row-segments (1D) are dilated, and then the set of column-segments (2D). See [58, 59] for more information. We compare this method, using both the DT-grid and the H-RLE data structure, to the optimized rebuild method in Listing 3. The results are shown in Figure 5.7(b). It might be surprising that the optimized rebuild is the slowest, and in addition seem to scale worse than the complex, sparse method in Listing 4. However, this is a result of bad cache coherence in the optimized method. Because the grid points in the coordinate list are not sorted, data access is bound to get more and more scattered. This effect is more pronounced for large grids.

5.8 CONCLUSION AND FUTURE WORK

This was a short introduction to some of the data structures used in this thesis. We hope that the reader has been given some intuition for how they perform with respect to memory footprint and speed.

The main message in this chapter is that it is clear that there is always a trade off between memory and speed. The reduction in memory footprint comes at the cost of slower access to grid points. For more information, and considerably more benchmarks, we direct the interested reader to the papers [41, 58, 59, 61].

5.8.1 A special note on blocked grids

The point being made about the trade-off between memory size and speed is asymptotically well motivated. However, for some size ranges the blocked grids seem to be an exception to the rule. If the acceleration grid holding pointers to the individual blocks fits well in memory, then the blocked grid retains fast random access without a severely large memory footprint. No benchmarks are yet published using blocked grids in LSM, but at least the DB-grid [52, 55] is under active development, and we expect data to be reported shortly.

Moreover, when random access is less important, the blocked grid can be combined with any of the other topology compressing schemes presented above producing new and interesting data-structures.

CHAPTER 6

GEODESIC DISTANCE

GEODESIC distance measures the length of closest paths in space. A *geodesic* is a generalization of a straight line to curved spaces (manifolds) and describes the shortest path between two points. Distances and geodesics can be used for many things: path planning, image analysis, the description of geometrical properties to name a few.

In this chapter we extend some of the ideas from article [64], which presented an efficient way to compute approximate distance functions, or distance maps. The preliminary results presented here are expository and references may not be complete. Still, we consider these findings interesting enough to be included.

We first show how to compute distance in a three dimensional Euclidean space and outline an approach applicable for n dimensions. Then a method for computing intrinsic distances in a manifold is proposed, which complements our work in [16]. We end the chapter by showing how to let information flow out from an interface along the characteristics of the implicit function using only minor modifications of the algorithm in [64].

6.1 INTRODUCTION

Geodesic distance in a manifold Ω is defined as the metric $d : \Omega \times \Omega \longrightarrow \mathbb{R}$ that measures the length of the minimal continuous path $\boldsymbol{\gamma}$ connecting two points $\boldsymbol{a}$ and $\boldsymbol{b}$ in Ω, i.e.

$$d(\boldsymbol{a}, \boldsymbol{b}) = \min_{\boldsymbol{\gamma} \subset \Omega} \int_{\boldsymbol{\gamma}} ds. \tag{6.1}$$

Different metrics on Ω can be considered by the choice of distance element ds. In article [64] we focus on the computation of *distance maps* $\phi(\boldsymbol{x})$ with respect to a source set $S \subset \Omega$, such that

$$\phi(\boldsymbol{x}) \stackrel{\text{def}}{=} d(\boldsymbol{x}, S), \qquad \boldsymbol{x} \in \Omega. \tag{6.2}$$

These maps, or distance transforms, holds distances from all points $\boldsymbol{x}$ in the domain Ω to the source S. See [23] or [7] for some early work in this area. Today, distance maps are often found as the viscosity solution to the eikonal equation

$$\|\nabla \phi(\boldsymbol{x})\| = 1. \tag{6.3}$$

This is a well known approach and several methods for efficiently computing $\phi(\boldsymbol{x})$ in different settings exist, see [44, 65, 75] and references therein.

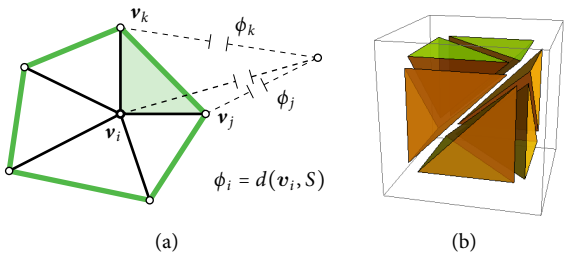


Figure 6.1: (a) The distance value at a vertex i is updated from all triangles T_{ijk} in the one-ring of i. The update value is given by the geometry of T_{ijk} and the distance values at vertices j and k. (b) In 3D a grid can be split into tileable tetrahedra. The grid points are located at the cube corners.

6.1.1 *Outline of the method*

Our method is not based on the common Hamilton-Jacobi discretization of the eikonal equation, instead we rely on the integral formulation in Equation (6.1). The method works by iteratively updating distance values for the vertices in a triangle mesh and striving for local optimality. More formally, the distance map is given as the solution to a dynamic programming problem (DPP), where each update is performed inside a triangle $T_{ijk} = [\boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{v}_k]$. We let the distance value at each vertex $\boldsymbol{v}_i$ with distance $\phi(\boldsymbol{v}_i) = \phi_i$ be updated based on the two other vertices $(\boldsymbol{v}_j, \boldsymbol{v}_k)$ and their distance values (ϕ_j, ϕ_k), see Figure 6.1(a). The method first initializes vertices close to the zero-set of the distance function called S. Then, an efficient algorithm propagates the solution outwards. For more information see [64].

6.2 GEODESIC DISTANCE IN THREE DIMENSIONS

As suggested in paper [64], it is straightforward to extend our method to higher dimensions. We briefly outline the process here. In 3D we start by assuming a tetrahedral mesh, which can be implicitly constructed from an underlying grid if needed. One such tessellation strategy is shown in Figure 6.1(b).

Following the notation in [64], we denote the local *boundary* for vertex i with Γ_i. It is shaded green in Figures 6.1(a) and 6.2. The boundary in 3D is the set of all triangles T_{ijk} in the generalized one-ring of $\boldsymbol{v}_i$. A particular update, given inside tetrahedra T_{ijkl}, has the face T_{jkl} as its corresponding part of Γ_i. In 3D the DPP from [64] reads

$$\phi_i = \min_{T_{jkl} \subset \Gamma_i} \Phi_{ijkl}, \qquad \text{for } i = 1, \ldots, n. \tag{6.4a}$$

$$\Phi_{ijkl} = \min_{\boldsymbol{v}^* \in T_{jkl}} \|\boldsymbol{v}_i - \boldsymbol{v}^*\|_2 + f(\boldsymbol{v}^*, \phi) \tag{6.4b}$$

$$\phi_i = 0, \qquad \text{for } i \text{ with } \boldsymbol{v}_i \in S. \tag{6.4c}$$

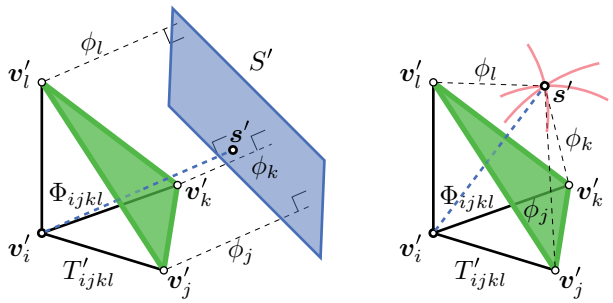


Figure 6.2: The minimizer is the closest distance from $\boldsymbol{v}_i$ to the source (dashed blue). (a) Using linear interpolation, *i.e.* approximating a planar source. (b) Using a point source interpolant and the sphere intersection solution. In 3D the boundary Γ of the tetrahedron T'_{ijkl} is the triangle T'_{jkl} (shaded green). A valid update is required to cross Γ.

We begin by expressing the interpolants proposed in [35, 71] in three dimensions. First, write a point on a triangular face of Γ_i in barycentric coordinates, as $\boldsymbol{v}^* = u\boldsymbol{v}_j + v\boldsymbol{v}_k + w\boldsymbol{v}_l$ with $u, v, w \geq 0$ and $u + v + w = 1$. The three dimensional counterpart of the linear interpolant from [35] then becomes

$$f_{linear}(u, v, w, \phi) = u\phi_j + v\phi_k + w\phi_l, \tag{6.5}$$

where ϕ_i, ϕ_j, and ϕ_k are the current distance values at node i, j, and k. Similarly, a 3D equivalent of the point source interpolant from [71] can be derived to be

$$\begin{aligned} f_{point}(u, v, w, \phi) &= \left\| (u\boldsymbol{v}_j + v\boldsymbol{v}_k + w\boldsymbol{v}_l) - \boldsymbol{s} \right\|_2 \\ &\vdots \\ &= \sqrt{\phi_j^2 u + \phi_k^2 v + \phi_l^2 w - \phi_{jk}^2 uv - \phi_{kl}^2 vw - \phi_{jl}^2 uw}, \\ &\text{for } u, v, w \in [0, 1], \text{ and } u + v + w = 1. \end{aligned} \tag{6.6}$$

This expression also makes use of the known inter vertex distances, $\phi_{ij} = \left\| \boldsymbol{v}_i - \boldsymbol{v}_j \right\|_2$.

The DPP in Equation (6.4) finds the position $\boldsymbol{v}^*$ that gives the shortest path from $\boldsymbol{v}_i$ to $\boldsymbol{s}$ by means of minimization. The length of this geodesic, $d(\boldsymbol{v}_i, \boldsymbol{s})$, is the update value. For this to be a valid update, we require the geodesic to be contained in the set formed by the two touching tetrahedra T_{ijkl} and T_{sjkl}. In Euclidean space this is equivalent to ensuring that the geodesic crosses the face T_{jkl} – the associated subset of the boundary Γ_i.

6.2.1 *The geometric construction*

Analogous to the two dimensional case presented in [64], the update value, Equation (6.4b), can also be found in a geometric setting. This is done by projecting the vertex in question on the source set S giving a position $\boldsymbol{s}$ and then measuring $d(\boldsymbol{v}_i, \boldsymbol{s})$. We do this in an isometric space. Let the tetrahedra T_{ijkl} be embedded as $T'_{ijkl} \in \mathbb{R}^3$. This can be achieved by first placing $\boldsymbol{v}'_i$ in $(0, 0)$ and letting $\boldsymbol{v}'_j = (d(\boldsymbol{v}_i, \boldsymbol{v}_j), 0)$. Then place $\boldsymbol{v}'_k$ such that $\left\|\boldsymbol{v}'_i - \boldsymbol{v}'_k\right\|_2 = d(\boldsymbol{v}_i, \boldsymbol{v}_k)$ and $\left\|\boldsymbol{v}'_j - \boldsymbol{v}'_k\right\|_2 = d(\boldsymbol{v}_j, \boldsymbol{v}_k)$. If T'_{ijk} is a valid triangle, then place $\boldsymbol{v}'_l$ such that $\left\|\boldsymbol{v}'_i - \boldsymbol{v}'_l\right\|_2 = d(\boldsymbol{v}_i, \boldsymbol{v}_k)$, $\left\|\boldsymbol{v}'_j - \boldsymbol{v}'_l\right\|_2 = d(\boldsymbol{v}_j, \boldsymbol{v}_l)$, and $\left\|\boldsymbol{v}'_k - \boldsymbol{v}'_l\right\|_2 = d(\boldsymbol{v}_k, \boldsymbol{v}_l)$. If the construction of any of the constituting triangles fail[1] or T'_{ijkl} does not form a valid tetrahedron, a Dijkstra update,

$$\Phi_{ijkl} = \min\left\{d(\boldsymbol{v}_i, \boldsymbol{v}_j) + \phi_j,\ d(\boldsymbol{v}_i, \boldsymbol{v}_k) + \phi_k,\ d(\boldsymbol{v}_i, \boldsymbol{v}_l) + \phi_l\right\}, \tag{6.7}$$

is used. Otherwise we can proceed and place $\boldsymbol{s}'$.

We consider two types of boundary conditions, or shapes of S. When using the linear interpolant (Equation 6.5) the corresponding source is a plane S'. The projection of $\boldsymbol{v}'_i$ on S' is shown in Figure 6.2(a). This gives the position $\boldsymbol{s}'$ which subsequently is used to measure $d(\boldsymbol{v}_i, \boldsymbol{s}')$. As previously mentioned it is required that $\boldsymbol{v}^*$ lies inside the triangular face T'_{jkl}.

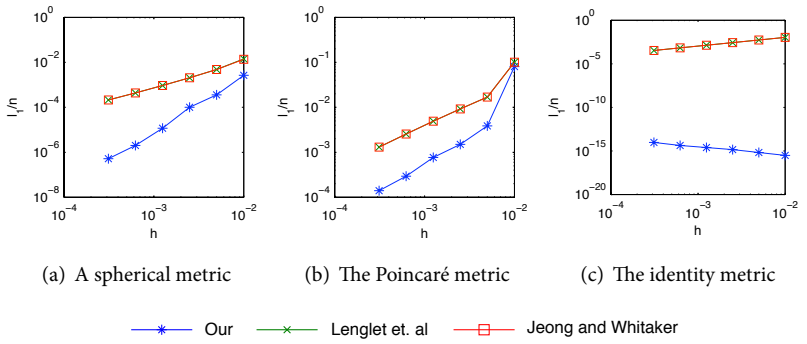
For the point source interpolant (Equation 6.6), the boundary is a single point $S' = \boldsymbol{s}'$. Its position is given by a three-sphere intersection, as illustrated in Figure 6.2(b).

6.2.2 *Preliminary results*

To verify the behavior of the 3D version of the algorithm we have performed a small convergence study using a proof of concept implementation. We partition the positive quadrant of the unit sphere into a regular grid and equip it with a metric tensor field, storing a metric tensor, $\boldsymbol{G}_i$, at each grid point. Distances are then computed with our method to a single point using the point source interpolant. We compare the results with a state of the art implementation [43] of the classical Hamilton-Jacobi discretization as originally proposed in [83]. We also compare to the fast marching type of method in [47].

The results, shown in Figure 6.3 together with the results from [64], indicate that our method behaves similarly in 2D and 3D. For Euclidean space our method is exact to numerical precision. Because of this the results are dominated by round-off errors and the l_1/n-norm actually grows under grid refinement, as demonstrated in 6.3(c). In anisotropic spaces our method consistently has a smaller error than competing work and seems to perform predictably. However, more tests are needed in order to fully characterize the three dimensional version of our method. The proof of concept

1. The edge lengths must pass the triangle inequality.



(a) A spherical metric (b) The Poincaré metric (c) The identity metric

Figure 6.3: We compute distances over part of the unit sphere and plot the mean of absolute error, l_1/n, against the grid step size h for different metrics.

implementation is furthermore relying on a complete tetrahedral tessellation. The extra burden of this explicit connectivity information in 3D makes the method less efficient, especially since the test is run on a regular grid. For the examples shown here our method is slower than competing work. We believe this can be helped by an implicit tessellation strategy that would significantly lower the memory footprint of the method and thus streamline processing.

Metrics and distances

For completeness, we list the test metrics and their corresponding distance functions here, the formulas appear courtesy of Anders Brun. The Poincaré metric:

$$\boldsymbol{G} = \boldsymbol{I}/(1 - x^2 - y^2 - z^2)^2, \tag{6.8}$$

with distance function

$$d(\boldsymbol{a}, \boldsymbol{b}) = \tfrac{1}{2} \cosh^{-1}\left(1 + \frac{2\|\boldsymbol{a}-\boldsymbol{b}\|_2^2}{(1-\|\boldsymbol{a}\|_2^2)(1-\|\boldsymbol{b}\|_2^2)}\right). \tag{6.9}$$

The sphere metric:

$$\boldsymbol{G} = \boldsymbol{J}^T\boldsymbol{J} \tag{6.10}$$

$$\boldsymbol{J} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \tau x & \tau y & \tau z \end{bmatrix} \tag{6.11}$$

$$\tau = -1/\sqrt{1 - x^2 - y^2 - z^2}, \tag{6.12}$$

with distance function

$$d(\boldsymbol{a}, \boldsymbol{b}) = \cos^{-1}\left(\boldsymbol{a}^T\boldsymbol{b} + \sqrt{(1-\boldsymbol{a}^T\boldsymbol{a})(1-\boldsymbol{b}^T\boldsymbol{b})}\right). \tag{6.13}$$

6.3 AN N-DIMENSIONAL UPDATE

We further formalize the geometric update by writing it as a general equation system in n dimensions, see [64] for the corresponding 2D situation. Assuming that the tessellation is given in terms of d-dimensional tetrahedra we have $d+1$ nodes in each simplex. For a vertex $\boldsymbol{v}_i$ we compute all tentative update values $\Phi_{ijk\ldots}$ and let ϕ_i be the minimum of these.

We do this by considering the boundary around vertex i (Γ_i) as the generalized one-ring of $\boldsymbol{v}_i$; this is a set of edges in 2D and a set of triangular faces in 3D. First, split Γ_i into simplices connected to $\boldsymbol{v}_i$ (triangles in 2D, tetrahedra in 3D) and consider each simplex in turn. Let the nodes of the simplex in question be labeled $\boldsymbol{v}_k$ with k from 1 to $d+1$, and let $\boldsymbol{v}_1$ be the node that we want to update. Each simplex that connects to $\boldsymbol{v}_i$ gives a single possible update $\Phi_{ijk\ldots}$ and the correct update value ϕ_i is the minimum of these.

To make the notation slightly easier to read we let the (single) tentative update $\Phi_{ijk\ldots}$ be written as simply Φ. The dependence of the vertices $\{\boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{v}_k, \ldots\}$ of the simplex is implicitly assumed. A linear tentative update value, Φ, is then computed as follows. The source S' is a hyper-plane determined by the expression $n_1x_1 + n_2x_2 + \cdots + n_dx_d + p = 0$. From projection we get the following equation system

$$\begin{cases} \hat{n}\cdot\boldsymbol{v}_1 + p = \Phi, \\ \hat{n}\cdot\boldsymbol{v}_k + p = \phi_k, & k = 2, 3, \ldots, d+1 \\ \|\hat{n}\|_2 = 1, \end{cases} \tag{6.14}$$

where $n_1, n_2, \ldots, n_d$, p, and Φ are $d+2$ unknown variables in $d+2$ equations. This allows to solve for the tentative update value Φ.

For the point source interpolant the system becomes nonlinear. Let the position of the point source be denoted by $\boldsymbol{s} = (x_1, x_2, \ldots, x_d)$. Using the equation of the circle (or n-sphere) we get the next system of d coupled equations

$$(\boldsymbol{v}_k - \boldsymbol{s})^2 = \phi_k^2, \qquad \text{for } k = 2, 3, \ldots, d+1. \tag{6.15}$$

This equation system has two real solutions if the tetrahedron connecting the boundary $T_{23\ldots}$ with the point $\boldsymbol{s}$ can be constructed. Then, the correct position of $\boldsymbol{s}$ can be determined as the solution of (6.15) giving the largest length in $d(\boldsymbol{v}_i, \boldsymbol{s})$ – the update value. If Equation (6.15) has no real solution a Dijkstra update is used.

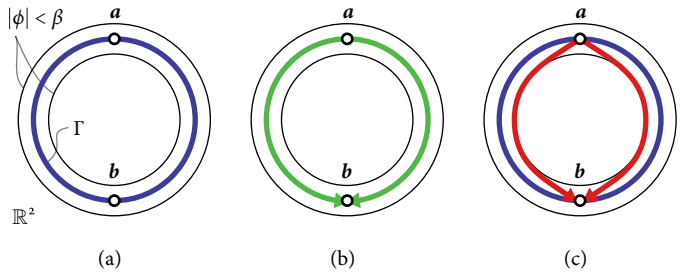


Figure 6.4: Intrinsic distance between two points in a manifold embedded in $\mathbb{R}^2$. (a) The interface (blue) in its narrow embedding. (b) The correct geodesics (green) follow the interface. (c) The approximate geodesics from [51] (red) are allowed to move freely over the narrow embedding, *i.e.* cut corners.

6.4 MEASURING INTRINSIC DISTANCE

The method we propose in article [64] can be used to measure distance on triangle meshes. When employing the implicit tessellation the method is equally applicable to LSM data on grids. In this case it can be used to, for example, re-initialize a signed distance function.

However, our method can also be used to measure *intrinsic* distance in manifold, that is distances along the interface, as we explain here. In [51] distances were computed over implicit surfaces by considering the intrinsic Hamilton-Jacobi equation

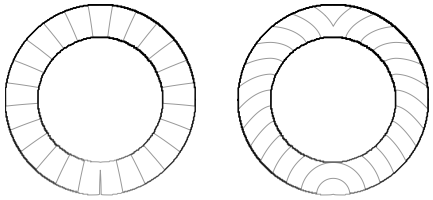
$$\|\nabla_S \psi(\boldsymbol{x})\| = 1, \tag{6.16}$$

where ∇_S measures the intrinsic, or projected, gradient. We let ψ be the distance function in the manifold to differentiate it from the distance function over the embedding (normally denoted ϕ). The method in [51] does, in fact, not use Equation (6.16) but rather employ the standard extrinsic gradient (Equation 6.3) and instead constrain distance computations in a small band around the interface. They show, under assumptions on the smoothness of the interface, that the extrinsic distances converge to the intrinsic ones under grid refinement. See Figure 6.4 for a depiction of the situation.

We can easily apply their technique with our algorithm and compute intrinsic distances. However, we can also see the embedding space as an anisotropic medium in which geodesics travel along level-sets only, *i.e.* the gradient of the level-set function ϕ should be orthogonal to the gradient of ψ. This is the same as to say

$$\nabla\phi(\boldsymbol{x}) \cdot \nabla\psi(\boldsymbol{x}) = 0. \tag{6.17}$$

This relation alone is not sufficient, in order to get the correct length of each geodesic we also need the relation in Equation (6.16). The following matrix defines a metric



(a) (b)

Figure 6.5: Level-curves of approximate geodesic distance functions defined over a narrow embedding. (a) The distance function is constrained to not change in the direction of the gradient using the metric from Equation (6.18). (b) Using the technique from [51] together with our distance method for point sources.

tensor which is approximately satisfies Equation (6.17) and also has unit cost along the interface:

$$\boldsymbol{G} = \left(\begin{bmatrix} (\frac{\partial \phi}{\partial x})^2 & \frac{\partial \phi}{\partial x}\frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial y}\frac{\partial \phi}{\partial x} & (\frac{\partial \phi}{\partial y})^2 \end{bmatrix} + \epsilon \boldsymbol{I} \right)^{-1}. \tag{6.18}$$

The addition of a small fraction of the identity matrix $\boldsymbol{I}$ ensures that the expression is invertible. In Figure 6.5 we show some examples illustrating the intended use. From a visual inspection our constrained approach gives geodesics that better adheres to the intrinsic distance function. Numerically, the data also suggest that our novel approach is valid. However, more work is needed before conclusive remarks can be made.

6.5 EXTRAPOLATION

If a quantity is only defined on the interface it can be "extended" outwards over the embedding by transporting it along the characteristics. The following Hamilton-Jacobi equation

$$\frac{\partial e}{\partial t} + \nabla e \cdot \hat{n} = 0, \tag{6.19}$$

can be used to extrapolate the scalar quantity e along $\vec{n}$. When run to steady state the quantity e does not change in the normal direction and the goal is thus achieved [27]. See Figure 6.6 for a visual illustration of the technique. In the static boundary formulation in [1], Adalsteinsson and Sethian instead rely on the static orthogonality property

$$\nabla e \cdot \hat{n} = 0, \tag{6.20}$$

to efficiently extrapolate quantities within their FMM algorithm.

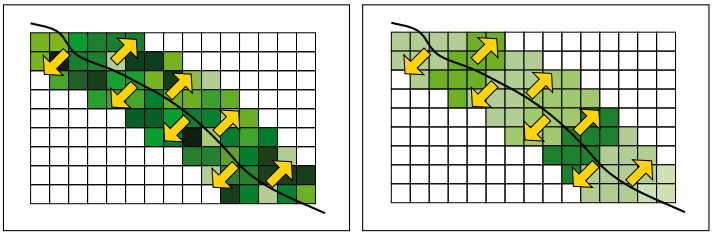


Figure 6.6: An illustration of how the extrapolation technique transports a quantity defined over the interface (black) outwards along the geodesics.

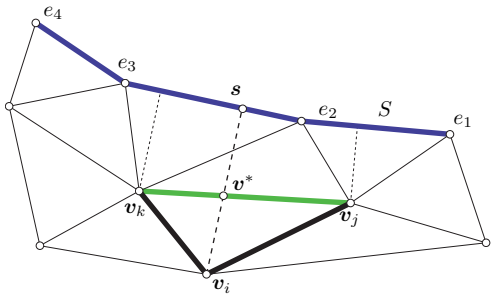


Figure 6.7: Extending a quantity out from a piecewise linear source S (blue). We seek the extension value e_i. From the relation $\nabla e \cdot \hat{n} = 0$ we see that $e_i = e(\boldsymbol{v}^*)$. With a linear scheme it is reasonable to assume that extension values vary linearly between $[\boldsymbol{v}_j, \boldsymbol{v}_k]$ and $e_i = f(t, e) = (1-t)e_j + t\,e_k$.

6.5.1 *Extrapolation based on interpolation*

In our proposed algorithm [64], each distance value is computed from two neighboring vertices in 2D, see Figure 6.1(a). An extension quantity can be transported within the algorithm by letting the nodes also carry an extra value. We let each vertex $\boldsymbol{v}_i$ be associated with an extension value e_i and assume that the nodes that lie on S are initiated. From the relation in Equation (6.20) we see that $e_i = e(\boldsymbol{s}) = e(\boldsymbol{v}^*)$. We extend the DPP in Equation system (6.4) with the following line

$$e_i = e(\boldsymbol{v}^*) = f(\boldsymbol{v}^*, e), \tag{6.21}$$

in order for the algorithm to also consider an extrapolated quantity.

When a linear interpolant is used, the distance at position $\boldsymbol{v}^*$ on the edge $[\boldsymbol{v}_j, \boldsymbol{v}_k]$ is given by the expression

$$f(\boldsymbol{v}^*, \phi) = f((1-t)\boldsymbol{v}_j + t\,\boldsymbol{v}_k, \phi) = (1-t)\phi_j + t\,\phi_k. \tag{6.22}$$

The corresponding extension value becomes

$$e_i = f(\boldsymbol{v}^*, e) = f((1-t)\boldsymbol{v}_j + t\,\boldsymbol{v}_k, e) = (1-t)e_j + t\,e_k. \tag{6.23}$$

For a geometric depiction see Figure 6.7 and 6.9(a).

The examples and discussion so far have assumed a linearly shaped source. With a point source on the other hand, multiple choices for how to interpret the situation are given.

First, there is the possibility to consider e to be angularly independent, that is $e_i = e(\boldsymbol{s})$. This is maybe most relevant in the case where we have several point sources. Then, the extension quantity can be used to, for example, differentiate between *Voronoi* regions.

The second possibility is to let the extension value be dependent on angle. To achieve this we assume that a small set of extension values around $\boldsymbol{s}$ are known, or can somehow be initialized[2]. We then extend these outwards, see Figure 6.8. In this scenario, we are also faced with the choice of interpolant. The original point source interpolant from [71] is not applicable. The linear interpolant is a possible prospect, but, when there is a dependence on angle a spherical linear interpolation seems more appropriate, see Figure 6.9(b). This interpolant also has the advantage of interpolating angles with numerical precision in the plane.

A modified algorithm

Every call to the update sub-routine in our algorithm identifies $\boldsymbol{v}^*$ as the intersection of the line segments $[\boldsymbol{v}_i, \boldsymbol{s}]$ and $[\boldsymbol{v}_j, \boldsymbol{v}_k]$, see Figure 6.7 and 6.8. This also implicitly gives the corresponding parameter t. If we let the causality of the extension values exactly follow that of the vertex distances it is straightforward to modify our algorithm

2. The exponential map around vertex i can be used to initialize the one-ring of i, see [28].

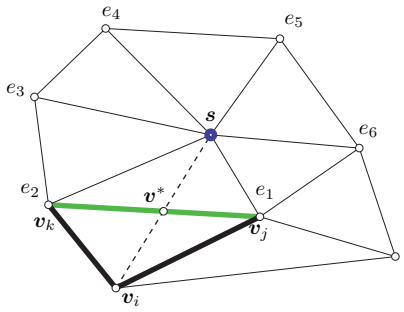


Figure 6.8: If the source is a point we consider two options. Either the point in question gets the extension value at $\boldsymbol{s}$, that is, $e_i = e(\boldsymbol{s})$. Or, we interpolate a set of known extension values around $\boldsymbol{s}$ outwards. Then, we get $e_i = f(\boldsymbol{v}^*, e)$. What type of interpolant f should be depends.

for extrapolation. With only minor changes of the original update algorithm in [64], we get

```
   Input: v_i, φ, e, f
1  foreach [v_j, v_k] ∈ Γ_i do
2      Place T'_ijk in ℝ² using inter-vertex distances
3      Use φ_j, φ_k, and f to also place s'
4      if s' can be placed and [v'_i, s'] crosses [v'_j, v'_k] then
5          Φ_ijk ← d(v'_i, s')
6          E_ijk ← f(v*, e)
7      else
8          if d(v_i, v_j) + φ_j < d(v_i, v_k) + φ_k then
9              Φ_ijk ← d(v_i, v_j) + φ_j
10             E_ijk ← e_j
11         else
12             Φ_ijk ← d(v_i, v_k) + φ_k
13             E_ijk ← e_k
14     if Φ_ijk < φ_i then
15         φ_i ← Φ_ijk
16         e_i ← E_ijk
17 return {φ_i, e_i}
```

Listing 6: A modified update sub-routine that also propagates extension values.

As in the original paper we let $d(\boldsymbol{x}, \boldsymbol{y})$ be the geodesic (shortest) distance between the two points $\boldsymbol{x}$ and $\boldsymbol{y}$. In Euclidean space $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$.

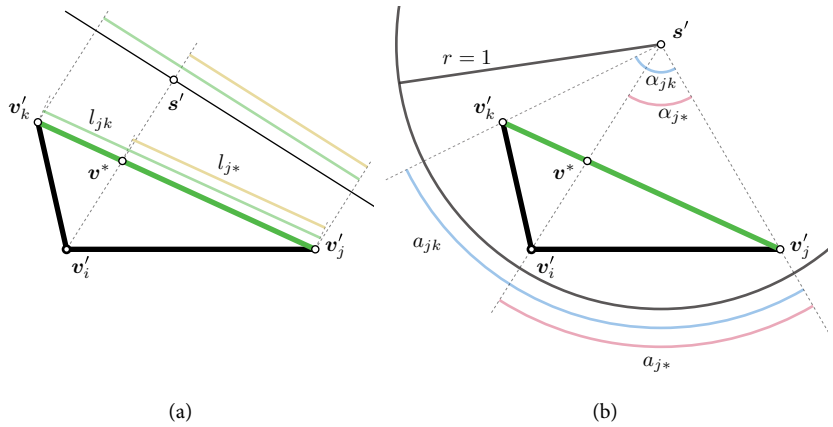


Figure 6.9: Interpolating extended values $f(\boldsymbol{v}^*, e)$ along the edge $[\boldsymbol{v}'_j, \boldsymbol{v}'_k]$. (a) When distance values are approximated with a linear interpolant it is reasonable to assume that the extended quantity also should be linearly interpolated. That is, $t = l_{j*}/l_{jk}$ and $f(t, e) = (1-t)\, e_j + t\, e_k$. (b) When the point source interpolant is used a spherical linear interpolant is often more appropriate. Then $t = a_{j*}/a_{jk} = \alpha_{j*}/\alpha_{jk}$ and $f(t, e) = (1-t)\, e_j + t\, e_k$.

CHAPTER 7

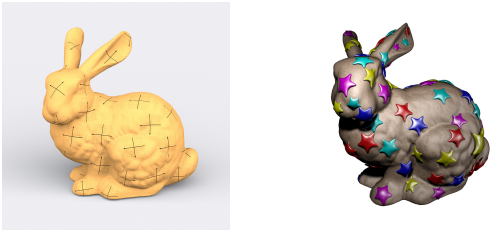
PARAMETERIZATION

TEXTURING of surfaces is an important technique in computer graphics. It works by mapping images onto geometry as shown in Figure 7.1(a). This allows geometry to appear more detailed than it actually is. In some cases it is impossible to model geometry at the same "apparent" resolution that a photograph mapped on a coarse geometry gives. At other times, geometry with too much detail makes practical operations painfully slow or impossible altogether. Thus, the texturing of surfaces in computer graphics has become an indispensable tool to compress data and increase realism.

Volumetric texturing is an extension of the texture mapping idea. Lately, interesting work in this field has been proposed, see for example [13, 70]. In these cases the idea is make a base surface seem more detailed by adding *geometric textures*. An example of this is shown in Figure 7.1(b).

7.1 INTRODUCTION

Much of the previous work, see for example the survey in [29] or the thorough introduction in [39], focus on creating a mapping which minimizes some specific error metric. Another well known, but "fixed", mapping is the exponential map, which uses geodesics as its coordinates. Because of the minimizing nature of geodesics the exponential map is well suited for parameterization. It can been used to parameterize



(a) (b)

Figure 7.1: Different forms of texture mapping. (a) An image of a coordinate system is mapped onto the bunny. (b) Volumetric, also called geometric, texturing. Sea star geometry is mapped onto the surface of the bunny, image courtesy of Anders Brodersen.

the one-ring around a vertex in a mesh [28, 38]. In [73] a method for computing the exponential map over a local surface neighborhood was proposed. In article [16] we propose a different numerical technique for approximating the exponential map, or rather, its inverse the *logarithmic map*. Our work displays similarity to [17] but has been significantly adapted to produce accurate mappings usable in computer graphics.

7.2 THE LOGMAP FRAMEWORK

The log map assigns coordinates to each point in space with respect to a base point $\boldsymbol{p}$ through geodesics. If an orthonormal basis is used these coordinates are called *Riemannian normal coordinates* (RNC).

In [17], see also the paper [16], it is shown that the logarithmic map, $\log_{\boldsymbol{p}}(\boldsymbol{x})$ can be computed from the gradient of a geodesic distance function

$$\log_{\boldsymbol{p}}(\boldsymbol{x}) = -\frac{1}{2}\nabla_{\boldsymbol{y}} d^2(\boldsymbol{x},\boldsymbol{y})\Big|_{\boldsymbol{y}=\boldsymbol{p}}. \tag{7.1}$$

when $\boldsymbol{x} \neq \boldsymbol{p}$. This is a well known expression, see e.g. [81]. Since $\|\nabla d(\boldsymbol{y},\boldsymbol{x})\| = 1$ and thus bounded, according to the eikonal equation (4.12), Equation 7.1 is well defined also in the limit when $\boldsymbol{x} = \boldsymbol{p}$. However, on the *cut locus* of $\boldsymbol{p}$ geodesics are not unique and the gradient of the distance function is undefined [69].

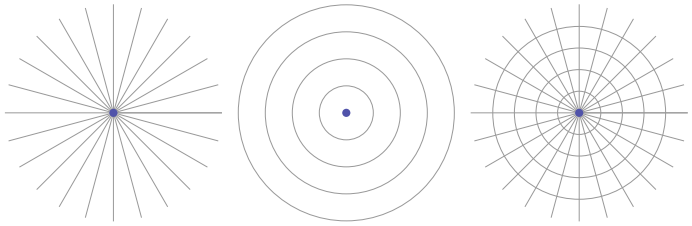
In order to obtain a discrete log map for a set of points in a mesh, the differentiation and distance calculation must be evaluated numerically. Some more intuition for the formula can be found by dividing the expression slightly

$$\log_{\boldsymbol{p}}(\boldsymbol{x}) = -\frac{1}{2}\nabla_{\boldsymbol{y}} d^2(\boldsymbol{y},\boldsymbol{x})\Big|_{\boldsymbol{y}=\boldsymbol{p}} = -\underbrace{d(\boldsymbol{x},\boldsymbol{y})}_{\text{distance}}\underbrace{\nabla_{\boldsymbol{y}} d(\boldsymbol{x},\boldsymbol{y})}_{\text{direction}}\Big|_{\boldsymbol{y}=\boldsymbol{p}}. \tag{7.2}$$

From this we see how the log map is built from geodesics. The coordinates are constructed from a direction, which in Euclidean space identifies a geodesic, and a distance along the geodesic in question. An analogy, the polar coordinate system, is depicted in Figure 7.2. Here, the geodesics are given by an angle (or direction) (a) and a distance (b).

To get some hands-on experience with the log map coordinates, we can test the formula analytically in the Euclidean plane. The distance function is $d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_2$ and we set $\boldsymbol{p} = \boldsymbol{0}$. This yields

$$\begin{aligned}
\log_{\boldsymbol{p}}(\boldsymbol{x}) &= -\frac{1}{2}\nabla_{\boldsymbol{y}}(\|\boldsymbol{x}-\boldsymbol{y}\|_2)^2\Big|_{\boldsymbol{y}=\boldsymbol{p}} \\
&= -\frac{1}{2}(2\boldsymbol{y}-2\boldsymbol{x})\Big|_{\boldsymbol{y}=\boldsymbol{p}} \\
&= \boldsymbol{x}-\boldsymbol{p} \\
&= \boldsymbol{x}.
\end{aligned}$$



(a) (b) (c)

Figure 7.2: (a) Iso-lines for angles with respect to the origin (blue). (b) Iso-lines of the distance to the origin. (c) Together these two "coordinates" comprise a polar coordinate system.

We see that the RNC given by the log map are unique (up to the choice of coordinate frame). The parameterization of the plane is moreover isometric, *i.e.* distortion free.

7.3 EFFICIENT COMPUTATIONS OF THE LOGARITHMIC MAP

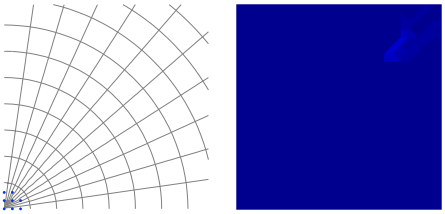
Paper [16] describes a generic approach to finding the logarithmic map in a manifold where distances can be computed. It is based on the averaging and differentiation of several distance functions and is shown to converge under sufficiently accurate distance estimates. We have used the method by Reimers [71] to compute distances. This method is superior when computing distances between points, see our discussion in [16], and also [64]. Reimers' method does not have a hard bound on its asymptotic run time complexity; but, in practice it scales with $O(m \cdot \log n)$ where n is the number of nodes and $m \approx n$.

In our method, the number of estimates to the distance function is constant and thus asymptotically optimal. However, the overall time is predominantly spent computing distance maps. Hence, there is still room for improvements, with a potentially big speed-up. In the examples in [16] we have used three, six, or more distance maps.

7.3.1 Parameterization by extrapolation

In Chapter 6 we show some preliminary results regarding distance computations in Euclidean and anisotropic spaces in two and three dimensions. Among other things we note how to use the proposed method in [64] to extrapolate values away from an interface by letting each geodesic transport a subordinate value along its path. That approach is now used as a parameterization technique, and we consider two different scenarios.

First we parameterize a surface with respect to a base point, $\boldsymbol{p}$. This is the classical setting and can be used to find local parameterizations used for decal compositing, *etc.*. When using an orthonormal basis the parameterization is given as a coordinate



(a) (b)

Figure 7.3: Using scalar extrapolation to parameterize the plane with respect to a base point (lower left). (a) Iso-lines of the coordinates (r, θ), where the angle is initialized from a small set of nodes (blue) close to the base point. (b) Color coding of the MIPS error metric. Blue – low distortion, to red – high distortion. In this example the accuracy of the distance component is on the order of 1e−16, for the angular component it is on the order of 1e−10.

system called Riemannian normal coordinates. The second scenario concerns the parameterization of the *embedding* of an interface. This coordinate system is, if orthonormal, called *Gaussian normal coordinates* (GNC). For this class uses such as the shell mapping in [70], or the geometric texturing in [12, 13], are applicable. Another interesting application was proposed in [15] where skeletons were extracted using a clever algorithm for computing GNC.

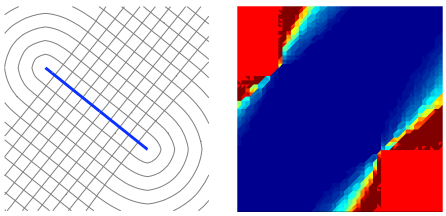
Both the coordinates systems above exist in arbitrary dimension and our examples readily translates as well. For more information on the, in differential geometry, well known Riemannian and Gaussian coordinate systems, see for example [85].

Parameterization around a point

Only minor alteration of the original method are needed in order to use the extrapolation technique introduced in Chapter 6 as a means to parameterize a surface around a point. We find an approximate log map in a polar coordinate system, (r, θ) centered in $\boldsymbol{p}$, by computing distances and setting $r(\boldsymbol{x}) = d(\boldsymbol{x}, \boldsymbol{p})$. The angles are simultaneously extrapolated from a small set of initiated vertices around $\boldsymbol{p}$. When using the spherical interpolant in the extrapolation of angles the accuracy is dominated by round-off errors, as seen in Figure 7.3. The RNC coordinates are given after a change to orthonormal basis.

Parameterization around an object

The GNC parameterizes the embedding around an interface Γ. Similarly to the point parameterization technique we approximate the GNC with a coordinate system, (s, t), where the first coordinate s is taken from the closest distance function $s(\boldsymbol{x}) = d(\boldsymbol{x}, \Gamma)$. The second parameter t is extrapolated within the distance computation. For each node the value is given by $t(\boldsymbol{x}) = t(\Gamma^*)$ meaning that the parameter value equals the



(a) (b)

Figure 7.4: Using scalar extrapolation to parameterize the embedding of a line. (a) Iso-lines of the coordinates (r, s). (b) Color coding of the MIPS error metric. Blue – low distortion, to red – high distortion.

Step size h	0.08	0.04	0.02	0.01
Angular error l_1/n	1.70e−03	5.83e−04	3.48e−04	1.42e−04

Table 7.1: The angular error under grid refinement.

parameter value on the closest point on the interface (Γ^*) as seen from $\boldsymbol{x}$. This requires t to be initialized on, or around, Γ. See Figure 7.4 for an example where the arc length is used to parameterize the curve Γ and then being extended over the embedding together with the distance function.

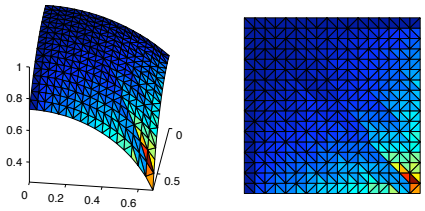
Since we consider Γ to describe an object, and not a point, the linear interpolant is used both for the distance component and for the transport of the extrapolation value (*e.g.* arc length).

7.4 EXAMPLES

In order to show some of the traits of the proposed parameterization method we now demonstrate some preliminary results. We begin by presenting a small convergence test. The RNC are approximated over the mesh of a partial sphere under refinement, for the initial setup see Figure 7.5(a). To measure accuracy we note the mean of the absolute error (l_1/n) of the angular component only in Table 7.1 [1]. The numbers presented in the table support the claim that the extrapolation method has the same linear convergence as the distance computations. For planar regions the angular component, like the distance, has numerical precision [71], see also Figure 7.3.

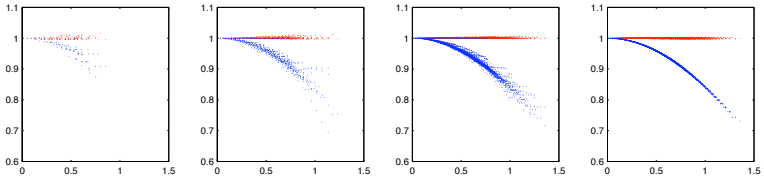
To illustrate properties of the parameterization we also plot the singular values of the mapping (see [16] or [39] for details) in Figure 7.6. The RNC give a mapping that is close to isometric in the vicinity of $\boldsymbol{p}$ and the distortion smoothly increases with

1. The distance coordinate is computed exactly as in [71], where it is shown to converge linearly.



(a) (b)

Figure 7.5: (a) A subset of the unit sphere such that $x, y \in [0, .7] \times [0, .7]$ and $z = \sqrt{1 - x^2 - y^2}$. The base point is set to be $\boldsymbol{p} = (0, 0, 1)$. (a,b) The approximated RNC is color coded with the MIPS metric, blue – low distortion, to red – high distortion. Note how the error grows with distance.



(a) $h = 0.08$ (b) $h = 0.04$ (c) $h = 0.02$ (d) $h = 0.02$, exact

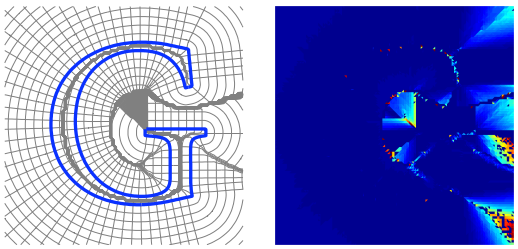
Figure 7.6: (a-c) The singular values (σ_1 – red, σ_2 – blue) of the approximated RNC on a partial sphere under mesh refinement. (d) The corresponding exact RNC sigmas at the resolution in (c).

distance, see also the color coding in Figure 7.5. This behavior is to be expected, and verified by the exact RNC in Figure 7.6(d). For more characteristics of the RNC see [16].

Figure 7.7 and 7.8 show the GNC of some realistic curves. In areas where the curve is smooth, the approximated GNC show little distortion, revealing our method to be a fast and attractive alternative to parameterize embeddings. As expected the parameterization is highly dependent on curvature. Areas where the level-curves are convex exhibit rarefaction, and conversely concave regions show compression.

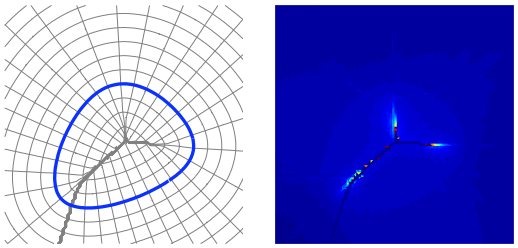
7.5 CONCLUSIONS AND FUTURE WORK

The parameterization technique introduced in this method seems to have great potential. It is extremely fast to compute, and yet retains the favourable traits of the RNC and GNC. In fact, real-time parameterizations are possible even for quite large neighborhoods.

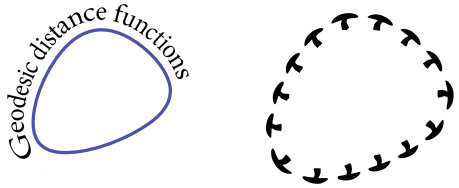


(a) (b)

Figure 7.7: Parameterization of the embedding space for the curve 'G'. (a) For smooth areas the distortion is low, but for some areas the GNC exhibit strong compression and rarefaction (b) In those regions the distortion is large in the MIPS metric.



(a) (b)



(c) (d)

Figure 7.8: Parameterization of the embedding space for a smooth closed curve. (a) The curve. (b) The GNC show low distortion except for at the medial axis according to the MIPS error metric. (c) Text is mapped onto the embedding space. (d) A mushroom texture is mapped repeatedly around the curve.

On the downside, the strong dependence on the curvature limits the method's usefulness too sufficiently smooth regions. However, filtering and averaging of distance maps produce better parameterizations in some situations, as indicated in [16]. We believe that these types of mappings can be used wherever fast local parameterizations are important. For example, in decal compositing [73], and geometric texturing [13, 70].

An important issue of the linear approximation method proposed here is seen, for example, in the bottom right region of the letter 'G' in Figure 7.7. This region has its closest point in a sharp convex corner of the letter, which is a *point*. This means that the dependent part of the distance function exhibits nonlinear behavior, and a linear interpolant is less suitable. A significant advance would be to combine the linear and point interpolant (together with extrapolation) to better handle such cases.

BIBLIOGRAPHY

[1] D. Adalsteinsson and J. Sethian. The fast construction of extension velocities in level set methods. *Journal of Computational Physics*, 148:2–22, 1998.

[2] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[3] P. Alexandroff. *Elementary Concepts of Topology*. Dover, Mineola, NY, 1961.

[4] J. D. Anderson. *Computational Fluid Dynamics*. McGraw-Hill Science/Engineering/Math, February 1995.

[5] M. Bardi and I. Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Birkhauser, 1997.

[6] J. Bloomenthal. *Introduction to Implicit Surfaces*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, first edition edition, August 1997.

[7] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344 – 371, 1986.

[8] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):173–192, 2001.

[9] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 28–36, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[10] R. E. Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, Stanford, CA, USA, 2003. Advisor-R. Fedkiw.

[11] T. Brochu and R. Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[12] A. Brodersen. *Flexible Methods for Geometric Texturing - From Terrain Visualization to Geometric Texture Mapping*. PhD thesis, Department of Computer Science, University of Aarhus, 2007.

[13] A. Brodersen, K. Museth, S. D. Porumbescu, and B. Budge. Geometric texturing using level sets. *IEEE Trans. Vis. Comput. Graph.*, 14(2):277–288, 2008.

[14] J. W. Bruce and P. J. Giblin. *Curves and Singularities*. Cambridge University Press, 1984.

[15] A. Brun. *Manifolds in Image Science and Visualization*. PhD thesis, Linköping University, 2007.

[16] A. Brun, O. Nilsson, M. Reimers, K. Museth, and H. Knutsson. Computing Riemannian normal coordinates on triangle meshes. *In submission*, 2009.

[17] A. Brun, C.-F. Westin, M. Herberthson, and H. Knutsson. Fast manifold learning based on riemannian normal coordinates. In *SCIA*, pages 920–929, 2005.

[18] T. F. Chan and L. A. Vese. Image segmentation using level sets and the piecewise-constant mumford-shah model. Technical report, Tech. Rep. 0014, Computational Applied Math Group, 2000.

[19] D. L. Chopp. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106(1):77–91, 1993///.

[20] R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM J. Res. Develop.*, 11:215–234, 1967.

[21] M. Crandall and P. Lions. Viscosity solutions of hamilton-jacobi equations. *Trans. of the American Mathematical Society*, (277), 1983.

[22] M. G. Crandall, L. C. Evans, and P. L. Lions. Some properties of viscosity solutions of hamilton-jacobi equations. *Transactions of the American Mathematical Society*, 282(2):487–502, 1984.

[23] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[24] M. P. do Carmo. *Riemannian geometry*. Birkhäuser, 1992.

[25] J. Dongarra, A. Lumsdaine, X. Niu, R. Pozo, and K. Remington. A sparse matrix library in C++ for high performance architectures. In *Proceedings of the Second Object Oriented Numerics Conference*, pages 214–218, 1994.

[26] L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, June 1998.

[27] R. P. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152(2):457–492, 1999.

[28] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Comput. Aided Geom. Des.*, 14(3):231–250, 1997.

[29] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[30] J. D. Foley. *Computer Graphics: Principles and Practice*. Addison Wesley, September 1995.

[31] N. Foster and R. Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[32] S. K. Godunov. A difference scheme for numerical solution of discontinuos solution of hydrodynamic equations. In *Math. Sbornik*, volume 47, pages 271–306, 1959.

[33] D. Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[34] R. Goldman. Curvature formulas for implicit curves and surfaces. *Comput. Aided Geom. Des.*, 22(7):632–658, 2005.

[35] R. Gonzalez and E. Rofman. On deterministic control problems: An approximation procedure for the optimal cost i. the stationary problem. *SIAM Journal on Control and Optimization*, 23(2):242–266, 1985.

[36] S. Gottlieb and C.-W. Shu. Total variation diminishing runge-kutta schemes. *Math. Comput.*, 67(221):73–85, 1998.

[37] A. Harten. High resolution schemes for hyperbolic conservation laws,. *Journal of Computational Physics*, 135(2):260 – 278, 1997 (1982).

[38] K. Hormann and G. Greiner. MIPS: An efficient global parametrization method. In P.-J. Laurent, P. Sablonnière, and L. L. Schumaker, editors, *Curve and Surface Design: Saint-Malo 1999*, Innovations in Applied Mathematics, pages 153–162. Vanderbilt University Press, Nashville, TN, 2000.

[39] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: theory and practice. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 1, New York, NY, USA, 2007. ACM.

[40] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Gigantic deformable surfaces. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 141, New York, NY, USA, 2005. ACM.

[41] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.*, 25(1):151–175, 2006.

[42] C. J. Isham. *Modern Differential Geometry for Physicists (World Scientific Lecture Notes in Physics)*. World Scientific Publishing Company, 1989.

[43] W.-K. Jeong, P. Fletcher, R. Tao, and R. Whitaker. Interactive visualization of volumetric white matter connectivity in dt-mri using a parallel-hardware hamilton-jacobi solver. In *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007)*, pages 1480–1487, 2007.

[44] W.-K. Jeong and R. Whitaker. A fast iterative method for eikonal equations (under review). *SIAM JSC*, 2007.

[45] G. Johansson, O. Nilsson, A. Söderström, and K. Museth. Distributed ray-tracing in an open source environment. In *SIGRAD 2006. The Annual SIGRAD Conference, Special Theme: Computer Games, November 22–23, 2006, Skövde, Sweden, Proceedings.*, volume 19, page 5. Linköping University Electronic Press, Linköpings universitet, 2006.

[46] D. E. Knuth. *Art of Computer Programming : Fundamental Algorithms*, volume 1. Addison-Wesley Professional, 3 edition, July 1997.

[47] C. Lenglet, E. Prados, J. Pons, R. Deriche, and O. Faugeras. Brain connectivity mapping using Riemannian geometry, control theory and PDEs. *SIAM Journal on Imaging Sciences*, 2008.

[48] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115:200–212, 1994.

[49] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure, 2004.

[50] G. H. Markstein. *Nonsteady Flame Propagation*. Macmillan, New York, 1964.

[51] F. Mémoli and G. Sapiro. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces: 730. *J. Comput. Phys.*, 173(2):764, 2001.

[52] K. Museth. An efficient level set toolkit for visual effects. In *SIGGRAPH '09: SIGGRAPH 2009: Talks*, pages 1–1, New York, NY, USA, 2009. ACM.

[53] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 330–338, New York, NY, USA, 2002. ACM.

[54] K. Museth, D. E. Breen, R. T. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *The International Journal of Eurographics Assoc.: Computer Graphics Forum*, 24(4):821–841, December 2005.

[55] K. Museth and M. Clive. Cracktastic: fast 3d fragmentation in "the mummy: Tomb of the dragon emperor". In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks*, pages 1–1, New York, NY, USA, 2008. ACM.

[56] K. Museth, M. Clive, and N. B. Zafar. Blobtacular: surfacing particle system in "pirates of the caribbean 3". In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 20, New York, NY, USA, 2007. ACM.

[57] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 721–728, New York, NY, USA, 2002. ACM.

[58] M. B. Nielsen. *Efficient and High Resolution Level Set Simulations, Data Structures, Algorithms and Applications*. PhD thesis, Aarhus University, 2007.

[59] M. B. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, pages 1–39, February 2006.

[60] M. B. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Virtually infinite resolution deformable surfaces. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 66, New York, NY, USA, 2006. ACM.

[61] M. B. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set methods. *ACM Trans. Graph.*, 26(4):16, 2007.

[62] O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis, an Eulerian approach with Lagrangian particle tracking. *Visualization, 2005. VIS 05. IEEE*, pages 407–414, 2005.

[63] O. Nilsson and A. Brun. Distance maps in an arbitrary metric tensor field – An iterative solver for 2-D charts. In *Proc. Swedish Symposium in Image Analysis*, 2008.

[64] O. Nilsson, M. Reimers, K. Museth, and A. Brun. Efficient computations of geodesic distance. *In submission*, 2009.

[65] S. Osher and R. Fedkiw. *Level Set and Dynamic Implicit Surfaces*. Springer-Verlag New York Inc., 2003.

[66] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[67] S. Osher and C.-W. Shu. High-order essentially nonsocillatory schemes for hamilton-jacobi equations. *SIAM Journal on Numerical Analysis*, 28(4):907–922, 1991.

[68] D. Peng, B. Merriman, S. Osher, H.-K. Zhao, and M. Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[69] X. Pennec. Intrinsic statistics on riemannian manifolds: Basic tools for geometric measurements. *J. Math. Imaging Vis.*, 25(1):127–154, 2006.

[70] S. D. Porumbescu, B. Budge, L. Feng, and K. I. Joy. Shell maps. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 626–633, New York, NY, USA, 2005. ACM.

[71] M. Reimers. *Topics in Mesh based Modelling*. PhD thesis, University of Oslo, September 2004.

[72] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, June 1992.

[73] R. Schmidt, C. Grimm, and B. Wyvill. Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph.*, 25(3):605–613, 2006.

[74] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci.*, volume 93, pages 1591–1595, 1996.

[75] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, June 1999.

[76] G. shan Jiang and D. Peng. Weighted ENO schemes for Hamilton-Jacobi equations. *SIAM J. Sci. Comput*, 21:2126–2143, 1997.

[77] G. shan Jiang, C.-W. Shu, and I. L. Efficient implementation of weighted ENO schemes. *J. Comput. Phys*, 126:202–228, 1995.

[78] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, August 1988.

[79] J. Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, 151(2):616–648, 1999.

[80] J. C. Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2 edition, 2004.

[81] R. Subbarao and P. Meer. Nonlinear mean shift over riemannian manifolds. *Int. J. Comput. Vision*, 84(1):1–20, 2009.

[82] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, September 1994.

[83] R. Tsai, H. Zhao, and S. Osher. Fast sweeping algorithms for a class of hamilton-jacobi equations. *SIAM journal on numerical analysis*, 41(2):673–694, 2004.

[84] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[85] R. M. Wald. *General Relativity*. University Of Chicago Press, June 1984.

[86] R. T. Whitaker. A level-set approach to 3d reconstruction from range data. *Int. J. Comput. Vision*, 29(3):203–231, 1998.

[87] C. Wojtan, N. Thürey, M. Gross, and G. Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3):1–10, 2009.

[88] H. Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation*, 74:603–627, 2004.