TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Analysis of flow and visibility on triangulated terrains

Citation for published version (APA):
Tsirogiannis, K. (2011). *Analysis of flow and visibility on triangulated terrains*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven.
https://doi.org/10.6100/IR717789

DOI:
10.6100/IR717789

Document status and date:
Published: 01/01/2011

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

Analysis of Flow and Visibility on Triangulated Terrains

Constantinos Perikleous Tsirogiannis

Analysis of Flow and Visibility on Triangulated Terrains

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 14 november 2011 om 16.00 uur

door

Constantinos Perikleous Tsirogiannis

geboren te Komotini, Griekenland

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. M.T. de Berg

Copromotor:
dr. H. J. Haverkort

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Tsirogiannis, Constantinos

Analysis of Flow and Visibility on Triangulated Terrains /
by Constantinos Perikleous Tsirogiannis.
Eindhoven: Technische Universiteit Eindhoven, 2011.
Proefschrift.
A catalogue record is available from
the Eindhoven University of Technology Library
ISBN: 978-90-386-2829-5
NUR 993

Subject headings:
computational geometry / algorithms / geographic information science
CR Subject Classification (1998): I.3.5, E.1, F.2.2

Promotor: prof.dr. M.T. de Berg
faculteit Wiskunde & Informatics
Technische Universiteit Eindhoven

copromotor: dr. Herman J. Haverkort
faculteit Wiskunde & Informatics
Technische Universiteit Eindhoven

Kerncommissie:
prof.dr. L. Arge (Aarhus University)
prof.dr. P. van Oosterom (Technische Universiteit Delft)
prof.dr. J. van Wijk (Technische Universiteit Eindhoven)

The work in this thesis is supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 639.023.301.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

© Constantinos P. Tsirogiannis. All rights are reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Cover Design: Constantinos P. Tsirogiannis with help from Paul Verspaget
Printing: Eindhoven University Press

As water spins in circles twice ...

Tiamat, $\Gamma\alpha\acute{\iota}\alpha$

Contents

Preface iii

1 Introduction 1
1.1 Flow Modelling on Digital Terrains . . . 2
1.1.1 Flow Modelling on DEMs . . . 4
1.1.2 Flow Modelling on TINs . . . 5
1.1.3 Flow on TINs According to the DSD Model . . . 6
1.2 Visibility on Triangulated Terrains . . . 10
1.3 Contributions of This Thesis . . . 13

2 An Efficient Mechanism for Routing Flow on TINs 15
2.1 Introduction . . . 15
2.2 Computing Information for a Single Trickle Path . . . 17
2.3 Expanding Multiple Paths Simultaneously . . . 21
2.4 Extracting Other Drainage Information . . . 29
2.5 Concluding Remarks . . . 33

3 Computing Drainage Structures on TINs: Practical Issues 35
3.1 Introduction . . . 35
3.2 Description of Implementation and Experiment Settings . . . 39
3.2.1 Implementation Using CGAL . . . 40
3.2.2 Experimental Set-Up . . . 41
3.3 The Complexity of Flow Structures . . . 42
3.3.1 The Complexity of Flow Paths . . . 42
3.3.2 The Complexity of Watersheds, Strip Maps and River Networks . . . 44
3.4 Quality of Inexact Flow Models . . . 47
3.5 An Output-Sensitive Algorithm for Computing Watershed Maps . 54
3.6 Concluding Remarks . . . 56

4 Identifying Watersheds on Noisy Terrains 57
4.1 Introduction . . . 57

4.2 Description of the Main Algorithms 60
4.2.1 Algorithms for Merging Watersheds 60
4.2.2 Algorithms for Matching Watersheds 63
4.3 Experimental Evaluation 65
4.3.1 A Robust Software Package for Computing Watersheds . . 65
4.3.2 Experimental Setup 66
4.3.3 Evaluation of the Algorithms 66
4.3.4 Discussion 68
4.3.5 Other Experiment Settings 69
4.3.6 Selecting a Unification Threshold 70
4.4 Conclusions 70

5 The Complexity of Visibility Maps on TINs Under Noise 73
5.1 Introduction 73
5.2 Visibility Maps Resulting from Perspective Projection 76
5.3 Terrains That Almost Satisfy the Assumptions 84
5.4 Concluding Remarks 88

6 Conclusions and Future Research 91

References 95

Summary 103

Curriculum Vitae 105

Preface

According to an ex-colleague, it is rather unfair that a PhD thesis bears the name of a single person. This is because several individuals may have contributed to the presented work; either directly or serving as an influence to the one that appears as the only author. In the next few lines I would like to thank most of those people that helped me in any way to carry out my PhD project.

First of all, I must express my gratitude for my two supervisors, Mark de Berg and Herman Haverkort. For the countless hours that you have spent for my supervision, for the many strange questions that you never neglected to answer, for making the present thesis possible, I honestly thank you. During these four years you have helped me a lot to structure my way of thinking, and to be more pragmatic and responsible in my work.

I would also like to thank my colleagues Amir Ali Khosravi, Peter Hachenberger, Bettina Speckmann, Alexander Wolff, Marcel Roeloffzen, Kevin Verbeek, Anne Driemel, Kevin and Maike Buchin, Marco Verstege, Fred van Nijnatten and the rest of the members of the Algorithms group, past and present, for the nice atmosphere that we had in the office during these four years. Thanks to Ignaz Rutter, Marcus Krug and Rodrigo Silveira for the joy of doing research together. Also, I want to thank the members of my thesis committee; professors Lars Arge, Peter Van Oosterom and Jack van Wijk for reading the thesis and providing insightful comments. I am really grateful to prof. Lars Arge and prof. Jens Christian Svenning for offering me a post doc position at Aarhus University, thus allowing me to continue doing research in a prestigious academic institute.

I am also grateful to the following people from the academic community for their help during the last years: Sylvain Pion, Monique Teillaud, Martijn van Leusen, Phillip Verhagen, Evanthia Papadopoulou and Panagiotis Rondogiannis. At this point I want to thank especially prof. Ioannis Emiris for helping me in my first steps in the world of Computational Geometry. Furthermore, I would like to thank here the Teachers who attracted my attention to Mathematics and Computer Science during my high school years: Nikolaos Fragkoudakis, Alekos Kostarelos, Dionysios Stamatis and Panagiotis Vigklas.

Thanks also go to my dear friends in Eindhoven: Nikos Sotiropoulos, Marietta Gontikaki, Chrysostomos Batistakis and the rest of the Greek graduate community of Eindhoven, Dina Ribena, Antonino Panteras and Mayla Barbarosa, Costas

Chatzikokolakis, Yan Shi, Dimitris Cheliotis, and Elias Tsigaridas and Constantinos Tsakalidis for helping me on my first days in Aarhus.

This thesis is dedicated to my family; my beloved brother Christos, and my parents Athena and Perikles. Thank you for your unconditional support all these years.

Chapter 1

Introduction

Through the course of history, human activity was always adapted to and influenced by the surrounding landscape. Cities were built close to water resources, such as rivers and lakes, which also meant they had to be adequately protected in fear of floods. Fortresses, temples and other monuments were built on hill tops and generally on prominent locations that provided high visibility of the nearby area. There is the ever-present need for man to use the landscape for his own advantage. This requires to extract information about different properties of the landscape surface; which part of a terrain is going to be covered by water in case of heavy rainfall? Which locations are visible from any point within a specific region? In the past, the study of terrain surfaces involved the use of topographic maps but also three-dimensional miniatures of landscapes [40, 85]. Such maps and models would serve as approximate representations of the original real-world surface and important decisions were taken based on these representations.
During the 20th century, major breakthroughs in computer science also had an impact in geographical studies. The development of efficient computer systems offered the opportunity to digitise geospatial data and process these data in an automated manner. This gave birth to the broad field of *Geographic Information Science* (GIS) [95]. For the study of landscapes, it became possible to represent terrain surfaces with digital models, the so-called *digital terrain models*. Among all the different digital terrain models known so far, the most popular one is the *Digital Elevation Model* (DEM). A DEM represents a terrain as a regular grid on the xy-plane in which each square grid cell is assigned a height value. Another popular digital terrain representation is the *Triangulated Irregular Network* (TIN), which is a two-dimensional triangulation where each triangulation vertex is assigned a height value—see Fig. 1.1 for an illustration of the two terrain models. Triangulated terrains have the advantage that they allow for non-uniform resolution when representing landscapes; for regions where the landscape is rough one can use many small triangles for a more detailed representation while few large triangles are enough for representing flat regions. TINs and DEMs have been

used to approximate terrain surfaces for a multitude of applications: computing shortest paths and proximity structures [5, 42, 53, 54, 63, 83], hydrological applications [14, 59, 79, 80, 88], computer graphics and flight simulations [8, 77], civil engineering [37, 76], environmental applications [15, 75], applications that stem from landscape archaeology [30, 49, 50, 51, 52] as well as image processing [13, 44, 86, 92]. Indeed, digital terrains can be generally seen as representations of xy-monotone functions and as such they can be used to model concepts other than real-world landscapes.

1.1 Flow Modelling on Digital Terrains

One of the most important scientific topics that involves the use of digital terrain models is modelling the flow of water on surfaces. Simulating the course of water on a digital terrain and likewise determining the conditions under which water can accumulate at certain locations is nowadays the standard method for predicting floods and taking measures to prevent them [1, 22]. To do that we need to model the behaviour of water on the digital surface. That is, we need to define a digital *flow model*. The standard flow model that is used in digital terrain analysis can be summarized in two very simple rules; (i) water always follows the directions of steepest descent (DSD) on a surface and (ii) the DSD is unique for any point on the surface of the terrain. From hereon we will refer to this model as the DSD model. Of course, this model can be directly applied only to continuous surfaces, which is not the case for DEMs. Also, even for continuous surfaces, this model does not predict precisely the flow behaviour of water. The DSD model is purely geometrical and as such it does not completely capture the behaviour on real terrains, where aspects such as soil type and land cover also play a role. Nevertheless, purely geometric flow models are considered to be a useful abstraction for performing flow analysis and the DSD model is the most natural geometric flow model. Thus, efficient and accurate algorithms for computing flow-related structures on a digital terrain $\mathcal{T}$ according to the DSD model are highly desirable.
The most basic algorithmic question relating to flow is to compute, for a given point p on a terrain, where the water from p drains to. In other words, we want to compute the *trickle path* of p: the path of steepest descent starting at p and ending at the local minimum to which the water from p drains. The set of all points whose trickle paths *reach* p constitute the *watershed* of p. The subdivision of a terrain induced by the watersheds of its local minima is the *watershed map* of the terrain. Paths that follow the direction of steepest descent or steepest ascent on a terrain surface, that is *gradient paths*, are also used for defining topological structures on TINs. Consider a terrain $\mathcal{T}$ and consider that we connect the saddle points of $\mathcal{T}$ with the local minima and local maxima of $\mathcal{T}$ by expanding paths of steepest ascent and steepest descent from the saddles. The graph G_{sn} whose vertices correspond to the critical points of $\mathcal{T}$ and whose edges correspond to the gradient paths between the critical points is the *surface network* of $\mathcal{T}$. Surface

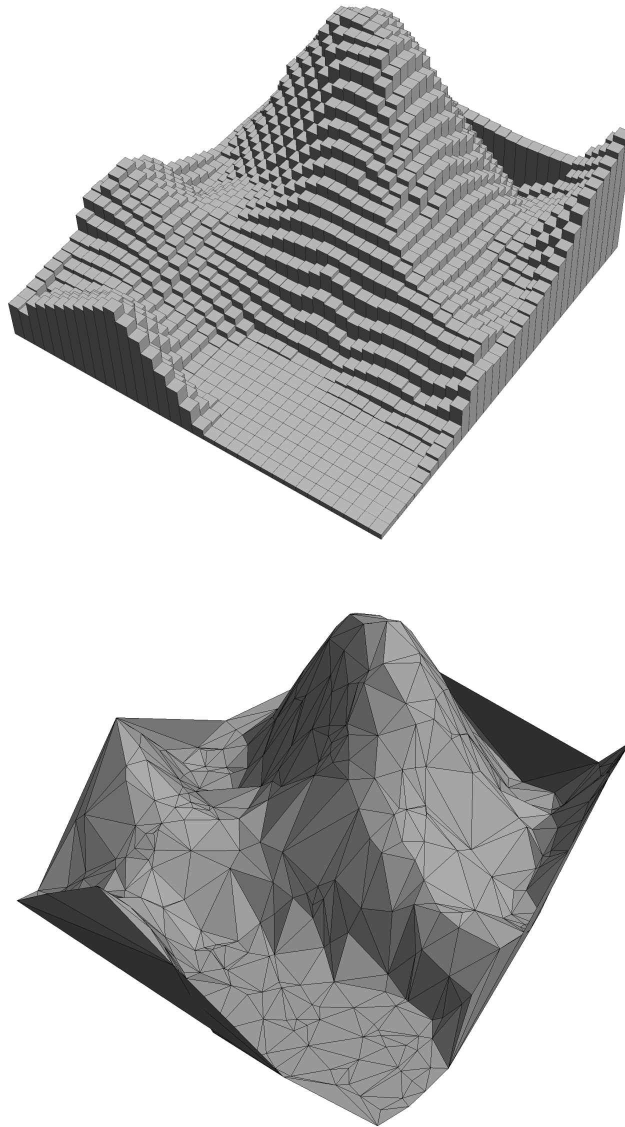


Figure 1.1 Examples of digital terrain models: (top) A Digital Elevation Model. (bottom) A Triangulated Irregular Network.

networks are considered to be useful abstractions of the topology of a terrain and play a key role in many applications [70, 73]. The computation of flow paths and watersheds on a digital terrain depends on the type of the terrain model used but

also on the way the flow model is applied.

1.1.1 Flow Modelling on DEMs

DEMs are used more often than any other digital model for simulating flow on terrains; this is because it is easy to design and implement algorithms on DEMs. For hydrological applications, since a DEM is a discrete non-continuous surface, it is not completely clear how to translate the DSD model to a DEM. Another drawback is that it is not always clear which locations on the DEM correspond to critical points; more than that, it is not straightforward how to delineate crisp characteristics of the terrain, like regions that have the morphology of ridges or channels.

One intuitive way to model flow on DEMs is to compute an approximation of a trickle path from each grid cell. A watershed of a cell c is then defined as a collection of grid cells whose approximate trickle paths reach c. The standard method to route flow on DEMs in this manner is the so-called *D8* method introduced by O' Callaghan and Mark [16]. This algorithm models flow in the following way. Consider a grid cell c in the interior of a DEM. If c is not a local minimum, then water flows from c to one of its (at most) eight neighbouring cells, the cell that has the smallest elevation. There are several variants of the $D8$ method [35, 74]; these variants were introduced because the standard method is notorious for computing flow paths and networks that have an unnatural shape [25]. For instance, Fairfield and Leymarie [35] introduced an improved version of this method in order to eliminate many parallel tributaries in the induced drainage networks.

Rather than computing for each grid cell the local minimum to which it drains, other methods for compute watersheds on DEMs by expanding the watershed boundaries from saddle points [78]. For such approaches it is important to first define a scheme for extracting critical points from the DEM [20, 93]. Then the watershed boundaries are expanded from the extracted saddle points as lines that approximate the direction of the up-hill gradient on the terrain surface.

Vincent and Soile [92] propose an algorithm for computing watersheds on DEMs by simulating a flooding process; according to the terminology used for geometric algorithms, their algorithm is a (bottom-to-top) space-sweep technique. Starting from the grid cell with the smallest elevation, the watersheds of the local minima on the terrain are delineated by symbolically tracking the terrain contour lines. These are the lines induced by the intersection of the sweep plane and the DEM surface. This technique is not restricted to grid terrains; it can also be applied to drainage networks that do not come from regularly shaped terrain models such as DEMs.

Other approaches even consider that the DSD is not unique for every point on the terrain [72, 25]. Such approaches aim to provide a more natural flow modelling for DEMs of coarser resolutions. However, methods of this kind may lead to instances where different local minima have overlapping watersheds and special care should be taken to avoid this.

Most flow-routing algorithms on DEMs use a scheme where flow is propagated only through a network rather than a two-dimensional surface. Recall that the surface of a DEM makes it difficult for a definition of flow that has a straightforward geometric interpretation. Several approaches try to alleviate this problem by first constructing another surface based on data extracted from the DEM, and then trying to extract the flow properties of the terrain from the constructed surface. Mitasova and Hofierka [62] present such an approach where they extract several attributes from DEMs by fitting an interpolation function to DEM points. Steger [84] uses smooth interpolation functions to delineate watershed boundaries on DEMs; his technique allows computing critical points and watershed boundaries on the terrain that are not restricted by the grid cell resolution.

1.1.2 Flow Modelling on TINs

For TINs the DSD model can be applied directly, since TINs are continuous surfaces (although here one also has to decide how to define flow on flat areas). The DSD model implies that a trickle path can extend through the interior of triangles and edges until ending at a local minimum. The edges that appear in the interior of a TIN $\mathcal{T}$ can be classified in three categories, depending on the DSD in the interior of the incident triangles. Fig. 1.2 illustrates these categories. An edge e is called a *transfluent* edge if the DSD in the interior of triangle incident to e points towards e, while for the other triangle the DSD points away from this edge. An edge e of $\mathcal{T}$ is called a *valley* edge or a *channel* if the DSD in the interior of both incident triangles points towards this edge. If for both incident triangles the DSD points away from e then this edge is called a *ridge* edge.

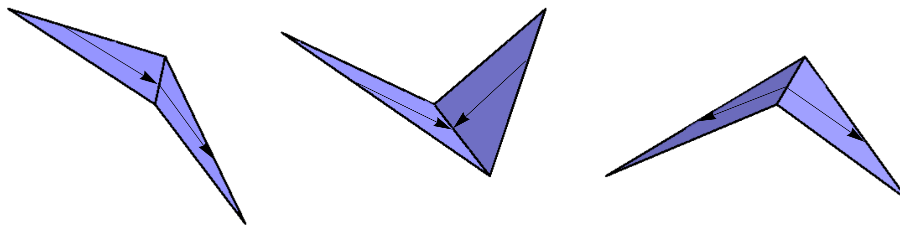


Figure 1.2 An illustration of the three different categories of TIN edges according to the configuration of the DSD: a transfluent edge (left), a valley edge (center) and a ridge edge (right).

Unfortunately, most approaches in the field of GIS do not follow the DSD assumption on TINs strictly. A common approach is to allow flow paths only follow the edges of the TIN, or the edges of a predefined network that involves the vertex set and specific triangle interior points [60, 68]. Restricting flow to the TIN edges makes the computations easier, but it does not lead to exact results. Also most of these

methods assign each triangle to a single watershed even when the DSD model implies that the triangle interior is crossed by a watershed boundary. Therefore, the computed watersheds appear as collections of full triangles.
Takahashi *et al.* [87] propose an algorithm that delineates watersheds on TINs by outlining their boundaries, that is by expanding ascending paths from the saddle points of the terrain. However, these ascending paths are approximated as sequences of terrain edges and do not follow the direction of steepest ascent on the terrain terrain surface strictly. This method is similar to flow-modelling approaches defined for DEMs such as the ones of Schneider [78] and Steger[84].

1.1.3 Flow on TINs According to the DSD Model

Jones *et al.* [47] are the first to consider the computation of drainage structures on TINs following strictly the direction of steepest descent. They provide a thorough description of how to calculate flow paths through triangle interiors and they propose simple algorithms for computing the areas that drain to valley edges, and for computing approximately the watershed map on a TIN. De Berg *et al.* [6] examine the worst-case *combinatorial complexity* of flow paths. The combinatorial complexity of a flow path is the number of segments that the path consists of. De Berg *et al.* proved that for a terrain $\mathcal{T}$ of n triangles the combinatorial complexity of a trickle path can be $\Theta(n^2)$ in the worst case. They showed that this is possible by providing a TIN that has a pyramid-like structure—see Fig. 1.3. On this terrain certain trickle paths cross $\Theta(n)$ triangles each $\Theta(n)$ times, yielding the proposed worst-case complexity. In fact, de Berg *et al.* primarily focus on proving the worst-case complexity of the *river network* of a TIN; that is the set of all points on the terrain whose watersheds are two-dimensional regions. Consequently, the river network of a TIN $\mathcal{T}$ consists of all valley edges of $\mathcal{T}$ plus the trickle paths that we get if we follow the DSD from the lowest vertex of each valley edge. By carefully positioning $\Theta(n)$ valley edges on the top of the proposed TIN, we get $\Theta(n)$ trickle paths where each such path belongs to the river network and the combinatorial complexity of each path is $\Theta(n^2)$.
The next step is to study the computation of watersheds in the DSD model. Yu *et al.* [96] examine the computational complexity of drainage-related queries on TINs. Given a TIN $\mathcal{T}$ they consider queries such as computing the area measure of the watershed of a point p on $\mathcal{T}$, or computing the subset of $\mathcal{T}$ (in fact of its river network) of terrain points whose watersheds have an area measure at least equal to a given value. They also introduce an algorithm for computing the watershed of a given point on $\mathcal{T}$. This algorithm, as well as the algorithms that support the rest of the queries that they consider, is based on a refined structure called a *strip*. This concept can be described as follows; suppose that we expand from each vertex of $\mathcal{T}$ all paths of locally steepest ascent and descent. These paths partition the TIN surface into strips, which together form the *strip map* of the TIN. The "bottom" of each strip is a segment of a valley edge, the "top" is a segment of a ridge edge—see Fig 1.4. The crucial property of the strips is that the trickle path from any point

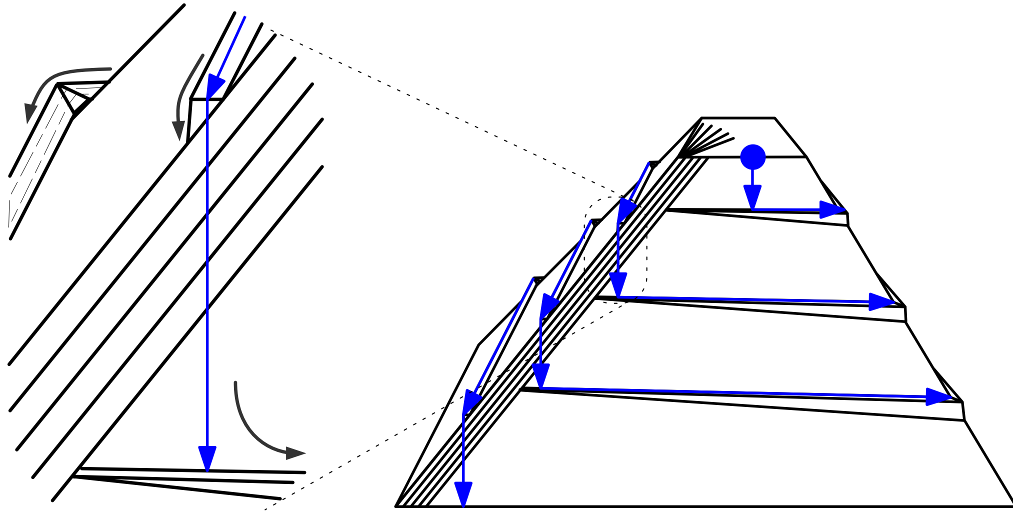


Figure 1.3 A construction that exhibits the worst-case complexity of a trickle path on a TIN. The presented terrain has a group of $\Theta(n)$ very skinny and long triangles at one corner. A network of "bridges" makes the depicted trickle path (designated by the arrows) to intersect the group of the skinny triangles $\Theta(n)$ times inducing $\Theta(n^2)$ intersections in total. The figure is based on an image that appears in the work of de Berg *et al.* [6].

in the interior of a strip leads to the valley edge at the bottom of the strip and, hence, drains to the same local minimum. Thus, for non-degenerate cases, the watershed of any local minimum is the union of one or more strips. However, in degenerate cases, the interior of a single watershed may not be connected [2]. Also, for a point p that is not a local minimum of $\mathcal{T}$, the watershed of p may not be the union of full strips [96]. McAllister [2], and McAllister and Snoeyink [3] examine several issues related to the properties of watersheds on TINs delineated according to the DSD model. They also describe an algorithm for computing watersheds on TINs that involves expanding paths of steepest descent/ascent only from a subset of the TIN vertices. However, it is not clear whether this algorithm always leads to

consistent results [7]. Using a construction similar to the adversarial TIN described by de Berg *et al.* [6], McAllister shows that the combinatorial complexity of the watershed map of a TIN can be $\Theta(n^3)$ in the worst case. The same tight bound applies for the combinatorial complexity of the strip map.

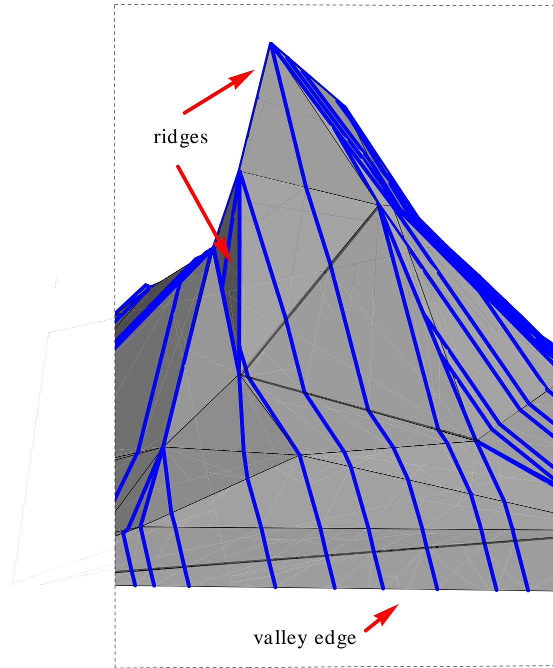


Figure 1.4 Part of the strip map of a TIN. The boundaries of the strips are indicated in blue colour.

The worst-case complexity of watersheds and flow networks in the DSD model seems quite unrealistic. In practice, it is very unlikely that a flow path will ever cross the same triangle more than once. In fact, the TINs that are used to induce the worst-case instances of the described drainage structures consist of extremely long and skinny triangles. Such TIN instances will probably never occur in real-world applications. De Berg *et al.* [7] study the worst-case complexity of drainage structures again, this time on so-called *realistic* TINs; that is TINs that fulfill a set of well-defined properties. Specifically for their analysis they consider TINs where the minimum angle of any triangle is at least some value α. They prove that the combinatorial complexity of a single trickle path on a terrain of this kind is $\Theta(n/\alpha^2)$ at worst case. The worst-case combinatorial complexity of the river

network and the watershed map on such terrains is $\Theta(n^2/\alpha^2)$. A quite remarkable result that they provide is that even for TINs whose xy-projections are Delaunay triangulations, the combinatorial complexity of watershed maps and river networks can still be $\Theta(n^3)$.

It becomes clear that, even for fat terrains, the worst-case complexity of drainage structures such as the watershed map or the strip map is quite high. Thus, the question arises: can we extract information on the drainage properties of TINs without computing these structures explicitly? For example, can we compute the exact area measure of the watershed of each local minimum on the terrain without computing the watersheds themselves? Since for general TINs the combinatorial complexity of trickle paths can be very high, can we extract information for one or more flow paths without actually constructing these paths? For instance, it is interesting to determine the local minima or TIN boundary points where a set of paths end, or to determine if these paths intersect certain regions of the terrain. This will be one of the topics studied in this thesis. Another interesting question that comes up is whether these worst case complexity bounds appear in practice; do such complex drainage structures occur in real-world data sets? Of course, this question can only be answered by developing software that computes watershed maps and flow paths on TINs following the DSD flow model strictly.

McAllister and Snoeyink [3], and Liu and Snoeyink [57] were the first to develop a software package that computes watersheds on TINs according to the DSD model. Liu and Snoeyink highlight an important issue that was overlooked by the previous theoretical approaches; drainage structures in the DSD model cannot be computed robustly using fixed-precision numbers. In particular, they consider the number of bits needed to compute exactly the intersection points of a trickle path with the TIN edges that it crosses. They show that this number grows linearly with the number of (transfluent) edges crossed by the path, potentially leading to large bit-sizes in the computations. In their implementation they use finite-precision arithmetic, and they observe that this may give inconsistent results, for example watersheds that do not contain exactly one local minimum. Thus, it becomes important to develop an implementation that uses exact arithmetic; that will show if it is actually efficient to compute drainage structures according to this flow model. This is another contribution of this thesis. If the exact computation of drainage structures requires numerical values of a very large bit-size then the exact DSD model can be used only for terrain data sets of relatively small size. This may be the case even when the computed drainage structures do not have a high combinatorial complexity.

Another important factor that affects the output of a flow model is the noise that appears in the input data. Noise may be caused by sampling with limited-accuracy equipment, data conversion between different terrain representation models, and calculations under fixed-precision arithmetic [94]. For a given TIN, noise can be represented by giving each vertex an interval of possible elevation values (rather than a single elevation value), while keeping the xy-coordinates of the vertices fixed. Thus we get an *imprecise* terrain: a terrain of which the elevations are not

entirely fixed [39]. A *realisation* or a *perturbation instance* of such an imprecise TIN is a terrain instance that results from selecting, for each vertex, a particular elevation value from its elevation interval. In theory, small perturbations of the elevations of the vertices of a TIN may induce substantial changes in its drainage structure. Driemel *et al.* [29] study the computational complexity of a variety of problems related to the computation of watersheds on imprecise terrains using either the exact DSD model on TINs or discretized methods of modelling flow. It is interesting to evaluate in practice to what extent the structure of the watersheds of a TIN changes if we perturb the elevation values of its vertices. Given different instances of the same TIN created by perturbing its vertices, how much of the terrain area appears as part of the same watershed among all different instances? Such an evaluation may provide insight as to how noise in the input data affects the output of software that is used for hydrological applications.

To be able to answer this question, we need to define what we consider to be the "same" watershed across different terrain realisations. As different perturbation instances of the same terrain lead to different watershed structures, it becomes challenging to identify which watersheds represent the same entity between those instances. For example, consider a TIN $\mathcal{T}$ and a local minimum p on the surface of $\mathcal{T}$. If we perturb the vertices of $\mathcal{T}$, the region that is covered by the watershed of p may change substantially, the region may become larger or smaller, or p may not even be a local minimum after the perturbation. Thus, what would be the best criterion to identify which watersheds correspond to each other before and after the perturbation? More formally, given the watershed maps of two or more realisations of the same terrain, what is the best way to decide which watersheds represent the same entity over all realisations? This is one more problem that we examine in this thesis.

To this point, there is a variety of works within the GIS community that study the impact of noise on the hydrological properties of a terrain model. Hebeler and Purves [43] examine the relation between the existence of noise and landscape morphology, and provide a case study on the changes among two neighbouring watersheds when noise is applied. Other works focus on the changes that appear on flow paths or the structure of the river network due to noise [33, 56]. The flow analysis which appears in these works refers only to DEMs. Even in this context, there has been no work so far that provides a well-defined algorithmic method for matching watersheds between different realisations of the same imprecise terrain.

1.2 Visibility on Triangulated Terrains

In the previous section, we provided an outline of several algorithmic approaches for computing flow structures on digital terrains. We saw that drainage structures on TINs may have a high combinatorial complexity in theory if water flow is modelled as following strictly the DSD on the terrain surface. There are also other important applications on TINs where structures of high combinatorial complexity

may occur. A good example are visibility applications.

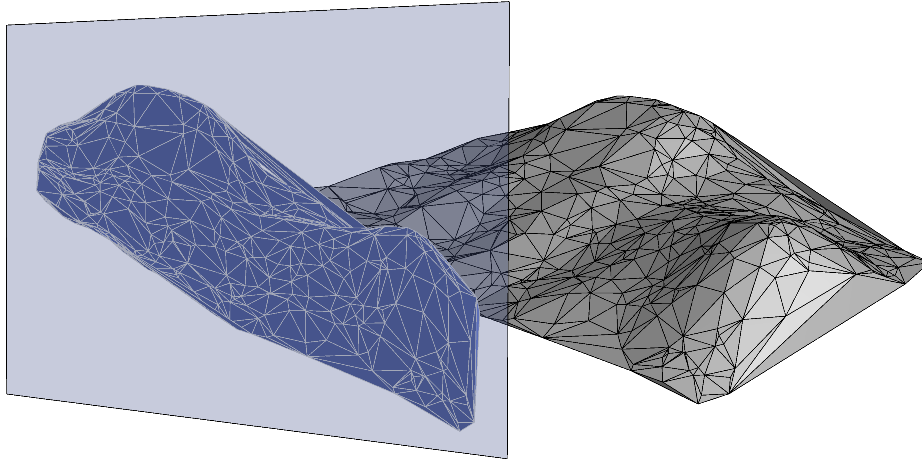


Figure 1.5 Example of a visibility map of a TIN induced by orthographic projection on a viewing plane orthogonal to the xy-plane.

Often it is desirable to compute which parts of a terrain $\mathcal{T}$ are visible from a given viewing point p_{view}. More precisely, for each triangle of $\mathcal{T}$, one wants to know exactly which parts are visible and which parts are invisible from p_{view}. The projections of the visible triangle parts onto a viewing plane form the so-called *visibility map* of $\mathcal{T}$ with respect to p_{view}. Visibility maps are useful for visualisation purposes; for example, they can be used for hidden-surface removal or shadow generation. There are several algorithms for computing visibility maps of terrains, the most efficient of which runs in time $O((n\alpha(n)+k)\log n)$ [48] where $\alpha(\cdot)$ is the inverse Ackermann function. Here n is the number of triangles in $\mathcal{T}$ and k is the output size. In other words, k is the complexity of the visibility map, which can be defined as the number of vertices[1] of the map. Each vertex of the map either corresponds to a triangle vertex, or to two edges whose projections onto the viewing plane intersect. In the worst case, $\Theta(n^2)$ pairs of edges have intersecting projections and all of these intersections are visible, so that the visibility map has complexity $\Theta(n^2)$. Such an example is provided in Fig. 1.6; there, a set of $\Theta(n)$ skinny obstacles in the foreground hides at parts a set of long horizontal triangles that appear in the background. The silhouette of the thin obstacles interacts with the long horizontal edges inducing $\Theta(n^2)$ vertices in the visibility map in total.
In most applications a quadratic complexity would make an explicit computation

[1] Formally, the complexity would be defined as the total number of vertices, edges, and faces of the map. In our setting this is always linear in the number of vertices, so we restrict ourselves to this quantity.

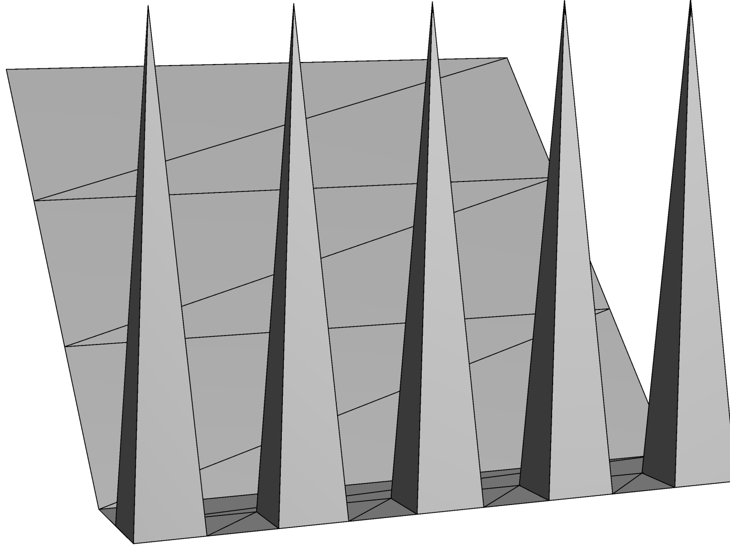


Figure 1.6 A construction that shows the worst-case complexity of the visibility map of a TIN. In this view, $\Theta(n)$ skinny obstacles in the foreground hide at parts $\Theta(n)$ long and almost horizontal edges in the background, inducing $\Theta(n^2)$ edge fragments in the visibility map.

of the visibility map infeasible. Fortunately such high complexity is seldom encountered. In fact, in practice it seems that the complexity of visibility maps is close to linear. Thus, it would be interesting to show why this structure usually has a low complexity in practical settings. Towards this direction, Moet *et al.* [66] studied the combinatorial complexity of visibility maps (but also of other terrain structures) on realistic terrains. Recall that a realistic TIN is a terrain that fulfills a set of desirable properties that are usually encountered in real-world data sets. Moet *et al.* examined visibility maps on TINs whose xy-domain is roughly a square, the triangles of the TIN are not arbitrarily skinny and the length of the longest TIN edge is not more than a constant factor greater than the length of the shortest edge. They proved that, in the worst case, the combinatorial complexity of the visibility map of such a terrain is $\Theta(n\sqrt{n})$. Yet, there is still a considerable gap between this worst-case bound and the linear complexity of visibility maps that is observed in real-world applications. Hence, it is still an open question why visibility maps of superlinear complexity do not appear in practice. This will be the final topic examined in the present work.

1.3 Contributions of This Thesis

In the following chapters of this thesis, we examine different techniques for handling high-complexity structures on TINs, both in theory and in practice. Chapters 2–4 deal with problems on the computation of drainage structures, while Chapter 5 is about the combinatorial complexity of visibility maps on TINs.

More specifically, in Chapter 2 we describe a general mechanism for extracting important information on drainage structures on TINs without computing these structures explicitly. Our contribution is a technique for tracing a collection of n paths of steepest descent on a terrain $\mathcal{T}$ implicitly in $O(n \log n)$ time. Recall that in the worst case, the combinatorial complexity of a single path of this kind is $O(n^2)$. The presented technique, rather than computing every possible intersection of these paths with the terrain edges, allows for computing information such as the exact points where each of the expanded paths end, but also other information on the topology of these paths. From this technique we derive $O(n \log n)$ time algorithms for: (i) computing, for each local minimum p of $\mathcal{T}$, the triangles contained in the watershed of p, and (ii) computing the surface network graph of $\mathcal{T}$. We also present an $O(n^2)$ time algorithm that computes the watershed area for each local minimum of $\mathcal{T}$. This chapter is based on joint work with de Berg and Haverkort which appeared in the 22nd ACM-SIAM Symposium on Discrete Algorithms [9].

Chapter 3 deals with problems that arise in practice when implementing algorithms for computing drainage structures on TINs following the DSD model strictly. As we discussed, the DSD model implies that water does not necessarily follow terrain edges, which makes designing exact algorithms difficult and causes robustness problems when implementing them. As a result, existing software implementations for computing watersheds are inexact: they either assume a simplified flow model or they perform computations using inexact arithmetic, which leads to inexact and sometimes inconsistent results. In this chapter we perform a detailed study of various issues concerning the exact or approximate computation of watersheds according to the DSD model; we provide the first implementation that computes watersheds on triangulated terrains exactly according to the DSD model, and we experimentally investigate its computational cost. Our experiments show that the algorithm cannot handle large data sets effectively, due to the bit-sizes needed in the exact computations and the computation of the strip map. Using our exact algorithm as a point of reference, we evaluate the quality of several existing inexact (but efficient) algorithms for computing watersheds. We also describe and theoretically analyse a new exact algorithm for computing watersheds, which avoids the computation of the strip map. This chapter is based on joint work with de Berg which will appear in the 19th ACM SIGSPATIAL Conference on Advances in Geographic Information Systems [12].

In Chapter 4 we consider the problem of identifying watersheds on imprecise TINs. Remember that computing watersheds on triangulated terrains in a robust manner is a difficult task also because it is sensitive to noise that appears in the elevation

values of the input. This is amplified by the existence of many small watersheds that obscure the overall hydrological structure of the terrain. In the present work we perform an experimental evaluation of various algorithms that may help alleviate these problems; we introduce and experimentally investigate algorithms for matching watersheds between different instances of a triangulated terrain that arise from adding noise to the elevations of the terrain model. These algorithms can be used to see which parts of a computed watershed map are reliable in the presence of noise. We also compare two methods for merging small watersheds into larger ones. We use these methods in combination with the watershed matching algorithms to assess which merging method is most effective in facilitating successful matching of watersheds. For the computation of watersheds on the examined TINs and the evaluation of the performance of the studied methods we used the robust software implementation that is presented in Chapter 3. This chapter is based on joint work with Haverkort which will appear in the 19th ACM SIGSPATIAL Conference on Advances in Geographic Information Systems [41].

In Chapter 5 we present an explanation for the low complexity of visibility maps on TINs which is observed in practice. In particular, we study the complexity of visibility maps of terrains whose triangles are fat, not too steep and have roughly the same size. The combinatorial complexity of a visibility map of such a terrain with n triangles is $\Theta(n^2)$ in the worst case. We prove that if the elevations of the vertices of the terrain are subject to uniform noise which is proportional to the edge lengths, then the worst-case expected (smoothed) complexity is only $\Theta(n)$. We also prove non-trivial bounds for the smoothed complexity of instances where some triangles do not satisfy the above properties. This chapter is based on joint work with de Berg and Haverkort that was published in the Journal of Computational Geometry [10]. Part of the results of this work were also published earlier in the 25th ACM Symposium on Computational Geometry.

Chapter 2

An Efficient Mechanism for Routing Flow on TINs

2.1 Introduction

Background and motivation. In many applications it is necessary to visualize, compute, or analyze flows on a height function defined over some 2- or higher-dimensional domain. Often the direction of flow is given by the gradient and the domain is a region in $\mathbb{R}^2$. The flow of water in mountainous regions is a typical example of this. Modelling and analyzing water flow is important for predicting floods, planning dams, and other water-management issues. Hence, flow modelling and analysis has received ample attention in the GIS community [36, 58, 68, 89].
In GIS, mountainous regions are usually modelled as a DEM or as a TIN. In computational geometry, a TIN is usually referred to as a *(polyhedral) terrain.* One advantage of polyhedral terrains over DEMs is that one can use a non-uniform resolution, using small triangles in rugged areas and larger triangles in flat areas. As we discussed in the introdution of this thesis, another advantage is that the surface defined by a polyhedral terrain is continuous, which makes flow modelling more natural. Indeed, the standard flow model on polyhedral terrains is simply that water follows the direction of steepest descent. To make the flow direction well defined, it is then often assumed—and we also make this assumption—that the direction of steepest descent is unique for every point on the terrain. For instance, the terrain should not contain horizontal triangles.[1]
In Chapter 1 we described several important structures related to the flow of water on a polyhedral terrain $\mathcal{T}$. The simplest structure is the path that water would follow starting from a given point p on the terrain. This path is called the *trickle*

[1] This can of course be ensured by a small perturbation of the elevations of the terrain vertices, but even small perturbations may have undesirable effects on the water flow. How to deal with horizontal triangles is therefore an important research topic in itself.

path of p and, as already mentioned, in our model it is simply the path of steepest descent. Another important structure is the *watershed* of a point p on $\mathcal{T}$, which is the set of all points on $\mathcal{T}$ from which water flows to p. In other words, it is the set of points whose trickle path contains p. Unfortunately, the combinatorial complexity of these structures can be quite high. For instance, de Berg *et al.* [6] showed that there are terrains of n triangles on which certain trickle paths cross $\Theta(n)$ triangles each $\Theta(n)$ times, resulting in a path of complexity $\Theta(n^2)$—see Fig. 1.3. McAllister [2] and McAllister and Snoeyink [3] showed that the total complexity of the watershed boundaries of all local minima can be $\Theta(n^3)$. By slightly modifying the construction provided by de Berg *et al.* we can in fact show that the boundary of a single watershed can have $\Theta(n^3)$ complexity. For *α-fat terrains*, where the angles of the terrain triangles are lower-bounded by a constant α, the situation is somewhat better: here the worst-case complexity of a single path of steepest ascent/descent is $\Theta(n/\alpha^2)$[7]. The complexity of a watershed, however, can still be $\Theta(n^2/\alpha^2)$.
It is not always necessary, however, to explicitly compute the structure of interest. For example, it may be sufficient to compute only the surface area of the watershed of a given local minimum, rather than an explicit description of the boundary of the watershed itself. The question thus arises: is it possible to compute the surface area of the watershed of a given local minimum without explicitly computing the watershed itself, thereby avoiding a worst-case running time of $\Theta(n^3)$?
A closely related structure on a terrain is the so-called *surface network* of $\mathcal{T}$. As defined in Chapter 1, this is the graph whose nodes are the critical points (local minima and maxima, and saddle points) of $\mathcal{T}$ and whose arcs are obtained by tracing paths of steepest ascent and descent from the saddle points to the local extrema [24, 70]. This graph has linear size, but explicitly tracing the paths of steepest ascent and descent from the saddle vertices results in a procedure that is very inefficient in the worst case. The surface network is related to the so-called Morse-Smale complex [61, 97], which has not only been used in GIS applications [24] but also for example in molecular shape analysis [18] (although here the domain is no longer in $\mathbb{R}^2$). The Morse-Smale complex has been originally defined for smooth surfaces, and in fact transferring the concept to the piecewise linear case—for example, to polyhedral terrains—is not straightforward. (The main difficulty lies in the fact that a path of steepest descent can intersect a path of steepest ascent.) Several methods have been proposed to define and compute Morse-Smale complexes on piecewise linear surfaces; see the paper by Čomić *et al.* [24] for an overview. In one way or another, these methods are always based on following certain paths of steepest descent/ascent. Sometimes an approximation is computed: the watershed of a point p (which is a cell of the so-called unstable Morse-Smale complex), for instance, would then be represented as the union of a certain subset of the terrain triangles. Existing algorithms of this type, however, are not exact: they are not guaranteed to find exactly those triangles for which all points have a trickle path containing p.

Our results. Inspired by the above, we study the problem of implicitly tracing paths of steepest descent or ascent on a polyhedral terrain $\mathcal{T}$ with n vertices. First, in Section 2.2, we give an $O(n \log n)$ algorithm that finds out where the trickle path of a given point p ends, without constructing the actual path (which would take $\Theta(n^2)$ time in the worst case). Our algorithm can also report all the triangles crossed by the path in the same amount of time. Then, in Section 2.3, we turn our attention to following multiple paths of steepest descent (or steepest ascent) simultaneously. We develop a mechanism for implicitly tracing n such paths in $O(n \log n)$ time in total. Using our mechanism, we can compute several of the flow-related structures mentioned above. In particular, in $O(n \log n)$ time we can:

- compute for each local minimum p of $\mathcal{T}$ the set of terrain triangles that lie completely in the watershed of p;
- compute the surface network of $\mathcal{T}$.

We also show how we can compute the exact surface area of all watersheds of $\mathcal{T}$ in $O(n^2)$ time.

Terminology and notation. In this chapter, for a terrain $\mathcal{T}$ we denote the set of its edges by E, and the set of its vertices by V. Edges in E are defined to be open, that is, they do not include their endpoints. For any point p we denote its z-coordinate by $z(p)$. For an edge $e \in E$ incident to a triangle t we call e an *out-edge* of t if e receives water from the interior of t through the direction of steepest descent. Otherwise we call e an *in-edge* of t. Thus, following from the definitions in the introduction of this thesis, e is a *valley* edge if e is an out-edge for both of its incident triangles, e is a *transfluent* edge if e is an out-edge for only one incident triangle, and e is a *ridge* edge if it is an in-edge for both of its incident triangles.

2.2 Computing Information for a Single Trickle Path

Let $\mathcal{T}$ be a terrain with n triangles, and let p be the point for which we want to compute the point where *trickle*(p) ends. As we only want to find where *trickle*(p) ends, we do not want to explicitly compute all intersection points between *trickle*(p) and the terrain edges. To avoid this, each time we encounter a sequence of edges that we crossed before, we jump to the first edge that we have not encountered so far. We can detect features that we already crossed, because we *mark* them the first time we hit them. Next we show how to do the above.

Define an *EV-sequence* to be the (ordered) sequence of terrain edges and vertices crossed by some path on $\mathcal{T}$. For a point $q \in$ *trickle*(p), let $\mathcal{S}(q)$ denote the

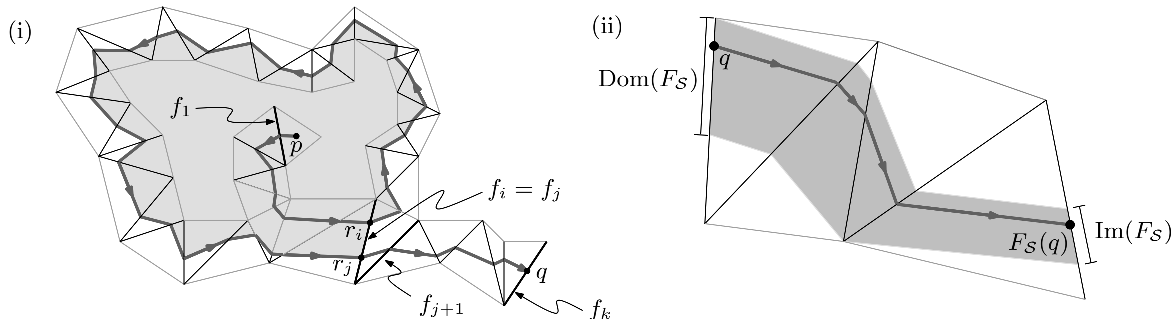


Figure 2.1 (i) The last cycle of the EV-sequence $\mathcal{S}(q)$ is $f_i, \ldots, f_j$, and the last chain is $f_{j+1}, \ldots, f_k$. (ii) The trickle function.

EV-sequence crossed by the part of *trickle*(p) from p to q. Consider a point $q \in$ *trickle*(p) and let $\mathcal{S}(q) = f_1 f_2 \cdots f_k$. Let j be the largest index such that the feature f_j occurs at least twice in $\mathcal{S}(q)$, and let i be the largest index with $i < j$ such that $f_i = f_j$. We call $f_i f_{i+1} \cdots f_j$ the *last cycle* of $\mathcal{S}(q)$, and we call $f_{j+1} \cdots f_k$ the *last chain* of $\mathcal{S}(q)$; see Fig. 2.1(i). We need the following lemma.

Lemma 2.1 *Let f be a feature in $\mathcal{S}(q)$ that only occurs before the last cycle of $\mathcal{S}(q)$. Then trickle(q) cannot cross f.*

Proof. Let $\mathcal{S}(q) = f_1, \ldots, f_k$ and let $f_i, \ldots, f_j$ be the last cycle of $\mathcal{S}(q)$. Let $e = f_i = f_j$ and let r_i and r_j be the intersection points of *trickle*(p) with e that correspond to f_i and f_j, respectively. Let $\pi(p, r_i)$ be the part of *trickle*(p) from p to r_i and let $\pi(r_i, r_j)$ be the part of *trickle*(p) between r_i and r_j. Note that *trickle*$(q) \subset$ *trickle*(r_j). Define $P := \pi(r_i, r_j) \cup \overline{r_i r_j}$. Then P is the boundary of a simple polygon—see Fig. 2.1(i), where this polygon is depicted in grey colour. Since trickle-paths cannot self-intersect and e can be crossed in only one direction by a trickle path, one of the paths $\pi(p, r_i)$ and *trickle*(r_j) lies completely inside P while the other lies completely outside P. This implies that a feature intersecting $\pi(p, r_i)$ can only intersect *trickle*(q) if that feature intersects $\pi(r_i, r_j)$ and, hence, occurs in the last cycle. □

Now imagine tracing *trickle*(p) and suppose we reach an edge e that we already crossed before. Let q be the point at which *trickle*(p) crosses e this time. After crossing e again, we may cross many more edges that we already encountered. Our goal is to skip these edges and immediately jump to the next new edge on the trickle path. By Lemma 2.1, the already crossed edges are either in the last cycle or in the last chain of $\mathcal{S}(q)$. In fact, since q lies on an edge crossed before, the last chain is empty and so the edges we need to skip are all in the last cycle. Therefore we store the last cycle in a data structure T_{cycle}—we call this structure the *cycle tree*—that allows us to jump to the next new edge by performing a query

FindExit(T_{cycle}, q). More precisely, if $\mathcal{C} = f_i, \ldots, f_k$ denotes the cycle stored in T_{cycle} and q is a point on f_i, then *FindExit*(T_{cycle}, q) reports a pair $(f_{\text{exit}}, q_{\text{exit}})$ such that f_{exit} is the first feature crossed by *trickle*(q) that is not one of the features in $\mathcal{C}$ and q_{exit} is the point where *trickle*(q) hits f_{exit}. The cycle tree stores the last cycle encountered so far in the trickle path, thus we have to update this tree according to the changes in the last cycle.
Besides the cycle tree we also maintain a list L which stores the last chain of $\mathcal{S}(q)$; these edges may have to be inserted into T_{cycle} later on. This leads to the following algorithm.

Algorithm *ExpandTricklePath*$(\mathcal{T}, p)$
Input: A triangulated terrain $\mathcal{T}$ and a point p on the surface of $\mathcal{T}$.
Output: The point where *trickle*(p) ends and the edges crossed by this path.

1. Initialize an empty cycle tree T_{cycle} and an empty list L, and set $q := p$. If q lies on a feature f, then insert f into L.
2. **while** q is not a local minimum and flow from q does not exit the terrain
3. **do** ▷ Invariant: T_{cycle} stores the last cycle of $\mathcal{S}(q)$, and L stores its last chain.
4. Let f be the first feature that *trickle*(q) crosses after leaving from q, and let q' be the point where *trickle*(q) hits f.
5. $q := q'$
6. **if** f is not marked
7. **then** Mark f and append f to L.
8. **else** Update T_{cycle} and empty L.
9. Set $(f_{\text{exit}}, q_{\text{exit}}) :=$ *FindExit*(T_{cycle}, q), mark f_{exit}, and set $q := q_{\text{exit}}$.
10. Append f_{exit} to L (which is currently empty) and update T_{cycle}.
11. **return** q.

It is easy to see that the invariant holds after step 1 and that it is maintained correctly, assuming T_{cycle} is updated correctly in steps 8 and 10. This implies the correctness of the algorithm. Next we describe how to implement the cycle tree.

Consider an EV-sequence $\mathcal{S}$ without cycles and assume that there is some trickle path that crosses the features in $\mathcal{S}$ in the given order. Let $\text{first}(\mathcal{S})$ denote the first feature of $\mathcal{S}$ and let $\text{last}(\mathcal{S})$ denote its last feature. We define the *trickle function* $F_{\mathcal{S}} : \text{first}(\mathcal{S}) \to \text{last}(\mathcal{S})$ of the sequence $\mathcal{S}$ as follows. If the trickle path of a point $q \in \text{first}(\mathcal{S})$ follows the sequence $\mathcal{S}$ all the way up to $\text{last}(\mathcal{S})$, then $F_{\mathcal{S}}(q)$ is the point on $\text{last}(\mathcal{S})$ where *trickle*(q) hits $\text{last}(\mathcal{S})$. If, on the other hand, *trickle*(q) exits $\mathcal{S}$ before reaching $\text{last}(\mathcal{S})$, then $F_{\mathcal{S}}(q)$ is undefined. We denote the domain of $F_{\mathcal{S}}$ (the part of $\text{first}(\mathcal{S})$ where $F_{\mathcal{S}}$ is defined) by $\text{Dom}(F_{\mathcal{S}})$, and we denote the image of $F_{\mathcal{S}}$ by $\text{Im}(F_{\mathcal{S}})$. Since we assumed there is a trickle path crossing $\mathcal{S}$, both $\text{Dom}(F_{\mathcal{S}})$ and $\text{Im}(F_{\mathcal{S}})$ are non-empty. Fig. 2.1(ii) illustrates these definitions. Note that $\text{Im}(F_{\mathcal{S}})$ is a single point when one of the features in $\mathcal{S}$ is a vertex. The following lemma follows from elementary geometry.

Lemma 2.2 *(i) The function* $F_{\mathcal{S}}(q)$ *is a linear function, and* $\text{Dom}(F_{\mathcal{S}})$ *and* $\text{Im}(F_{\mathcal{S}})$ *are intervals of* $\text{first}(\mathcal{S})$ *and* $\text{last}(\mathcal{S})$*, respectively. (ii) Suppose an EV-sequence* $\mathcal{S}$

is the concatenation of EV-sequences $\mathcal{S}_1$ and $\mathcal{S}_2$. Then $F_{\mathcal{S}}$ can be computed from $F_{\mathcal{S}_1}$ and $F_{\mathcal{S}_2}$ in $O(1)$ time.

Now consider an EV-sequence $\mathcal{S}(q) = f_1 \cdots f_k$ and let $\mathcal{C} = f_i, \ldots, f_j$ be the last cycle of $\mathcal{S}(q)$. The cycle tree T_{cycle} for $\mathcal{C}$ is a balanced binary tree, defined as follows.

- The leaves of T_{cycle} store the features $f_i, \ldots, f_{j-1}$ in order.
- For an internal node ν, let $lc[\nu]$ and $rc[\nu]$ denote its left and right child, respectively. Let $\mathcal{S}[\nu]$ denote the subsequence of $\mathcal{C}$ consisting of the features stored in the leaves below ν. Furthermore, let $\text{first}[\nu]$ and $\text{last}[\nu]$ denote the features stored in the leftmost and rightmost leaf below ν, respectively. Then ν stores the trickle function $F_{\mathcal{S}[\nu]}$, and the trickle function $F_{\mathcal{S}'[\nu]}$, where $\mathcal{S}'[\nu]$ is the sequence $f_\nu f'_\nu$ with $f_\nu = \text{last}[lc[\nu]]$ and $f'_\nu = \text{first}[rc[\nu]]$.

Lemma 2.3 *The function FindExit(T_{cycle}, q) can be implemented to run in $O(\log |\mathcal{C}|)$ time, where $|\mathcal{C}|$ is the length of the cycle stored in T_{cycle}.*

Proof. Imagine following *trickle*(q), starting at f_i, the first feature in $\mathcal{C}$. We will cross a number of features of $\mathcal{C}$, until we exit the cycle. (We must exit the cycle before returning to f_i again, because a trickle path cannot cross the same sequence twice without encountering another feature in between [6].) Let f^* be the feature of $\mathcal{C}$ that we cross just before exiting. We can find f^* in $O(\log |\mathcal{C}|)$ time by descending down T_{cycle} as follows.
Suppose we arrive at a node ν; initially ν is the root of T_{cycle}. We will maintain the invariant that f^* is stored in a leaf below ν. We will make sure that we have the point q_ν where *trickle*(q) crosses $\text{first}[\nu]$ available; initially $q_\nu = q$. When ν is a leaf we have found f^*, otherwise we have to decide in which subtree to recurse. The feature f^* is stored in the right subtree of an internal node ν if and only if

(i) $q_\nu \in \text{Dom}(F_{\mathcal{S}[lc[\nu]]})$, which means *trickle*$(q_\nu)$ completely crosses $\mathcal{S}[lc[\nu]]$, and

(ii) $F_{\mathcal{S}[lc[\nu]]}(q_\nu) \in \text{Dom}(F_{\mathcal{S}'[\nu]})$, meaning *trickle*$(q_\nu)$ reaches $\text{first}[rc[\nu]]$ after crossing $\mathcal{S}[lc[\nu]]$.

If these two conditions are met, we set $\nu := rc[\nu]$ and $q_\nu := F_{\mathcal{S}'[\nu]} \circ F_{\mathcal{S}[\nu]}(q_\nu)$, otherwise we set $\nu := lc[\nu]$.
Once we have found f^* and the point q^* where *trickle*(q) crosses f^*, we can compute the exit edge e_{exit} and point q_{exit} by inspecting the relevant triangle t incident to f^*: we just have to compute where the path of steepest descent from q^* exits t. □

It remains to explain how to update T_{cycle}. First consider step 8 of *ExpandTrickle-Path*. Suppose that, just before q reaches f, we have $\mathcal{S}(q) = f_1 \cdots f_k$. Let $f_i \cdots f_j$ be the last cycle of $\mathcal{S}(q)$ (which is stored in T_{cycle}) and $f_{j+1} \cdots f_k$ its last chain (which is stored in L). We know that f has been crossed before. By Lemma 2.1 this implies $f = f_m$ for some $m \geqslant i$. We distinguish two cases.

- If $m > j$, then f occurs in the last chain and, hence, in L. Now after crossing f the last cycle becomes $f_m \cdots f_k f$. So updating T_{cycle} amounts to first emptying T_{cycle}, and then constructing a new cycle tree on $f_m \cdots f_k f$, which can be done by a bottom-up procedure in $O(|L|)$ time.

- If $i \leqslant m \leqslant j$ then f occurs in the last cycle. Then after crossing f the last cycle becomes $f_m \cdots f_j f_{j+1} \cdots f_k f$. (In the special case that $m = j$, we in fact have $f_i = f_j = f$ and the last cycle becomes $f_j f_{j+1} \cdots f_k f$.) We can now update T_{cycle} by deleting the features $f_1 \cdots f_{m-1}$, and inserting the features $f_{j+1} \cdots f_k$. (Recall that the last feature of a cycle is not stored in the cycle tree.) Inserting and deleting elements from an augmented balanced binary tree T_{cycle} can be done in logarithmic time in a standard manner.

Next consider the updating of T_{cycle} in step 10. Let $f_i \cdots f_j$ be the last cycle before step 9, where we jump to the first new feature crossed by the trickle path. Let f_m be the last feature we cross before we exit the cycle, that is, the feature f^* in the proof of Lemma 2.3. Then after the jump, the last cycle becomes $f_m \cdots f_{j-1} f_i \cdots f_m$. (Essentially, the cycle does not change, but its starting feature changes.) Thus, to update T_{cycle} we have to split T_{cycle} between f_{m-1} and f_m into two cycle trees T^1_{cycle} and T^2_{cycle}, then merge these cycles trees again but this time in the opposite order (that is, putting T^1_{cycle} to the right of T^2_{cycle} instead of to its left). Splitting and merging can be done in logarithmic time, if we use a suitable underlying tree such as a red-black tree. We obtain the following theorem.

Theorem 2.4 *Let $\mathcal{T}$ be a terrain with n triangles and let p a point on the surface of $\mathcal{T}$. Algorithm ExpandTricklePath$(\mathcal{T}, p)$ traces the trickle path of p in time $O(n \log |C_{\max}|)$, where $|C_{\max}|$ is the length of the longest cycle in the EV-sequence of trickle(p).*

2.3 Expanding Multiple Paths Simultaneously

Our main interest is in designing an efficient algorithm that can expand a collection of $\Theta(n)$ paths simultaneously. Our next step towards this direction is to present how we can expand a collection of paths that emanate from the same point efficiently. We therefore design a subroutine that expands implicitly *upnet*(p), the *up-network* of a terrain point p; this is the set of all points on $\mathcal{T}$ reachable by a path of locally steepest ascent from p. Here the directions of locally steepest ascent are defined as follows. For a point $q \in \mathcal{T}$, let $\mathcal{B}_\epsilon(q)$ be the ball of infinitesimal radius centered at q. Let $\mathcal{M}_\epsilon$ be the set of points of locally maximum elevation in $\mathcal{B}_\epsilon(q) \cap \mathcal{T}$ whose elevation is greater than $z(q)$. Then the *directions of locally steepest ascent* at q are given by the vectors from q to each point in $\mathcal{M}_\epsilon$. We are interested in tracing the up-network implicitly since it plays a key role in the construction of the watershed of a given point [2]. We examine this issue in more detail in Section 2.4.

Next we describe our subroutine that expands *upnet*(p). We assume that the point p for which we want to compute the up-network is a terrain vertex [2]. An up-network is not necessarily a path; it can split and rejoin at terrain vertices. If we remove all terrain vertices from *upnet*(p), as well as all points that lie on a ridge edge, then *upnet*(p) is broken into several components which we call *up-paths*. We want our subroutine to compute the local maxima and/or the points at the boundary of $\mathcal{T}$ where *upnet*(p) ends.

Our algorithm is a space-sweep algorithm. Let h_z be the horizontal plane at elevation z and let P_z denote the set of up-paths intersecting h_z. We will maintain P_z as we move h_z upwards from p, meanwhile marking all the edges and triangles crossed by any of the up-paths. The difficulty in doing so is that an edge can be crossed by many up-paths and moreover that a single up-path can cross an edge many times.

To overcome these problems we proceed as follows. Let $\text{top}(\pi)$ denote the point up to which we have traced an up-path $\pi \in P_z$; the point $\text{top}(\pi)$ lies on or above h_z, and it will always lie on an edge. We associate π with the edge on which $\text{top}(\pi)$ lies. We denote the set of up-paths associated with an edge e when the sweep plane is at elevation z by $P_z(e)$. Let $P_z(e) = \pi_1, \ldots, \pi_k$; here and in the rest of the chapter we number the up-paths in $P_z(e)$ in increasing order of the z-coordinate of their tops. During the algorithm we will maintain each set $P_z(e)$ in an augmented tree according to this order. How this *bundle tree* is implemented will be discussed later. The idea is now to jump with each π_i to the first point where it crosses a terrain feature that lies completely above h_z. This feature can be either an edge or a vertex and we call it the *exit feature* of π_i. There can be several up-paths in $P_z(e)$ with the same exit edge. We call the collection of all such up-paths a *bundle* and we will make sure that we can jump with an entire bundle to the common exit edge. To facilitate the jumping, we store the edges currently intersecting h_z in a data structure similar to the cycle tree of the previous section. We call our new structure a *contour structure* and we denote it by D_{contour}. Later we will explain how to implement D_{contour}, but first we return to the overall algorithm.

We define an order on the terrain vertices and edges, that specifies the order in which they are handled. Let *rank*(v), the *rank of a vertex* v, be the z-coordinate of v, and let *rank*(e), the *rank of an edge* e, be the z-coordinate of the lower endpoint of e. This implies that when we jump from an edge e, we jump to the first feature with rank greater than the elevation of h_z. For two features f_1, f_2 we define $f_1 \prec f_2$ if either $\mathit{rank}(f_1) < \mathit{rank}(f_2)$, or f_1 is a vertex and f_2 is an edge and $\mathit{rank}(f_1) = \mathit{rank}(f_2)$. We extend this partial order to a total order in an arbitrary manner. An event queue will store vertices and edges in $\prec$-order. The global algorithm is now as follows. (When we write “insert this feature into Q” we actually first check whether the feature is already present in Q and only do the insertion when this is not the case.)

Algorithm *ExpandUpNetwork*($\mathcal{T}, p$)
Input: A triangulated terrain $\mathcal{T}$ and a vertex p of $\mathcal{T}$.

[2] If this is not the case we can just add p in V and re-triangulate the terrain.

Output: The local maxima/boundary points on $\mathcal{T}$ where *upnet*(p) ends and the edges crossed by *upnet*(p).

1. Set $z := z(p)$, initialize D_{contour} with all edges intersecting h_z, and create an event queue Q storing only p.
2. **while** Q is not empty
3. **do** Remove from Q the feature f that is minimal in the $\prec$-order.
4. Set $z := rank(f)$ and update D_{contour}.
5. **if** f is a vertex, v
6. **then if** v is a local maximum
7. **then** output v.
8. **else** ▷ Expand v:
9. For each up-path π starting at v, let e_π be the first edge hit by π. If e_π is incident to v then report e_π, and insert the other vertex w of this edge into Q. If e_π is not incident to v, then add π to $P(e_\pi)$, insert e_π into Q, and mark and report e_π.
10. **if** f is an edge, e
11. **then if** e is an edge on the boundary of $\mathcal{T}$
12. **then** Output the tops of the paths stored in $P_z(e)$.
13. **else** ▷ Jump from e:
14. Split $P_z(e)$ into bundles. For each bundle b, proceed as follows: Let $f_{\text{exit}}(b)$ be the first feature crossed by b that lies completely above the sweep plane h_z. Mark and report any unmarked edges crossed by b. Insert $f_{\text{exit}}(b)$ into Q, and if $f_{\text{exit}}(b)$ is an edge then add b to $P_z(f_{\text{exit}}(b))$.

The correctness of the algorithm can be seen as follows. By induction we can argue that all up-paths are created. When we trace the first link of an up-path (step 9) we mark the crossed edge, and when we extend an up-path as part of a bundle (step 14) we mark all newly crossed edges. Furthermore, an up-path continues to be extended until it ends. Hence, all edges crossed by *upnet*(p) are marked and all reached local maxima and boundary points are reported if the steps are implemented correctly.

Before we explain the various steps of the algorithm in more detail, we discuss some properties of the paths and bundles generated by the algorithm. We start with the next basic lemma.

Lemma 2.5 *Let e_{in} and e_{out} be an in-edge and an out-edge, respectively, of a terrain triangle t. Let $p, q \in e_{\text{out}}$ with $z(p) > z(q)$, and let $p', q' \in e_{\text{in}}$ be such that $\overline{p'p}$ and $\overline{q'q}$ are parallel to the direction of steepest descent. Then $z(q') > z(p')$ if and only if the highest vertex of e_{out} is the lowest vertex of e_{in}.*

Proof. Since $z(p) > z(q)$ we know that p lies closer than q to the vertex incident to e_{out} with the highest elevation. Let v' be this vertex.
Let v be the vertex incident to e_{out} and e_{in}. We consider two cases:

$v = v'$: Since $\overline{pp'}$ and $\overline{qq'}$ are parallel, $dist(p, v) < dist(q, v)$ implies that $dist(p', v) <$

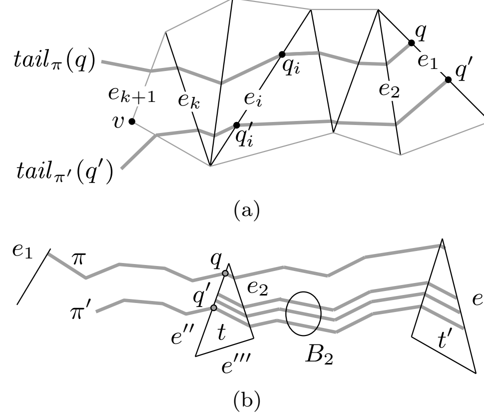


Figure 2.2 (a) Illustration for the proof of Lemma 2.6. (b) Illustration for the proof of Lemma 2.7.

$dist(q', v)$. But then we can only have $z(q') > z(p')$ if and only if v is the lowest vertex of e_{in}.

$v \neq v'$: Now q lies closer to v on e_{out} and, as $\overline{pp'}$ and $\overline{qq'}$ are parallel, point q' lies closer to v on e_{in} than p'. Let v'' be the other vertex incident to e_{in}. v'' has higher elevation than v' otherwise v and v'' are the vertices of lowest elevation in t and e_{in} cannot be an in-edge. On e_{in}, the point p' lies closer to v'' than q' so p' has a higher elevation than q'.

□

Consider a point q on an up-path π. We denote the part of π up to q by $tail_\pi(q)$. We define $rank(tail_\pi(q))$ to be the maximum rank of any edge crossed by $tail_\pi(q)$.

Lemma 2.6 *Let π and π' be two up-paths that cross the same transfluent edge e, and let q and q' be the points where they cross e. If $rank(tail_\pi(q)) > rank(tail_{\pi'}(q'))$ then $z(q) > z(q')$.*

Proof. Assume for a contradiction that $z(q') \geqslant z(q)$. Imagine tracing $tail_\pi(q)$ and $tail_{\pi'}(q')$ downwards as long as they follow the same EV-sequence. Let $\mathcal{S} = e_1, \ldots, e_k$ be this EV-sequence. Note that $e_1 = e$. Let q_i and q'_i denote the points where $tail_\pi(q)$ and $tail_{\pi'}(q')$ cross e_i, respectively—see Fig. 2.2(a).
Consider two consecutive edges e_i and e_{i+1}. Then the lowest vertex incident to e_i cannot be the highest vertex incident to e_{i+1}. Otherwise, e_i has a higher rank than any other edge following it, contradicting $rank(tail_\pi(q)) > rank(tail_{\pi'}(q'))$. The assumption $z(q') > z(q)$ thus implies, by Lemma 2.5 that $z(q'_i) > z(q_i)$ for all $1 \leqslant i \leqslant k$.

Let t be the triangle entered by $tail_\pi(q)$ and $tail_{\pi'}(q')$ before crossing e_k, and let v be the vertex of t not incident to e_k. We assume for simplicity that neither $tail_\pi(q)$ nor $tail_\pi(q')$ crosses v; adapting the argument is straightforward. Let e_{k+1} be the edge of t incident to the two lowest vertices of t. Note that v is one of these two vertices. Since $z(q'_k) > z(q_k)$, we know that $tail_\pi(q)$ crosses e_{k+1}. Let q_{k+1} denote the point where this crossing takes place. Since the endpoints of e_{k+1} are the two lowest vertices of t, either e_k or e'_{k+1} (the third edge of t) lies strictly above the interior points of e_{k+1}. But then any edge crossed by $tail_\pi(q_{k+1})$ has a lower rank than either e_k or e'_{k+1}, and the latter two edges are crossed by $tail_{\pi'}(q')$. Hence, $rank(tail_{\pi'}(q')) \geqslant rank(tail_\pi(q))$, and we reach a contradiction. □

Lemma 2.6 is used to prove that bundles cannot interleave, so that splitting a set $P_z(e)$ into bundles and adding these bundles to the sets $P_z(f_{\text{exit}})$ of their respective exit features can be done efficiently. Next we make this non-interleaving property precise.

Suppose that the algorithm jumps from an edge e in step 14. Note that two or more bundles in $P_z(e)$ may first follow the same edge sequence for some time before they split. For an edge e', we denote by $B_{\mathcal{S}}(e, e')$ the set of bundles that follow the same edge-sequence $\mathcal{S}$ from e to e' when $P_z(e)$ is processed. We call $B_{\mathcal{S}}(e, e')$ a *multi-bundle*. The tops of the up-paths when they reach e' after traversing $\mathcal{S}$ are called the tops of the multi-bundle.

Lemma 2.7 *(i) Let b be a bundle of $P_z(e)$. Then the paths in b are* consecutive *in $P_z(e)$. (ii) Let $B_1 := B_{\mathcal{S}_1}(e_1, e')$ and $B_2 := B_{\mathcal{S}_2}(e_2, e')$ be two multi-bundles crossing the same transfluent edge e'. Then B_1 and B_2 do not interleave on e', that is, there is a point on e' separating the tops of B_1 from the tops of B_2.*

Proof. To prove part (i), let π_1 and π_2 be the two outermost up-paths in b. Since up-paths don't cross, any up-path starting in between π_1 and π_2 follows the same edge-sequence as π_1 and π_2 up to $f_{\text{exit}}(b)$ and, hence, is an up-path in b.

To prove part (ii), assume without loss of generality that e_1 was handled before e_2. Thus $rank(e_1) \leqslant rank(e_2)$. We will show that no up-path $\pi \in B_1$ can separate B_2, that is, $\text{top}(\pi)$ cannot lie in between the tops of the outermost paths π_1 and π_2 in B_2. Showing that no up-path in B_2 can separate B_1 can be done in a similar, yet not symmetric, way.

If $rank(e_1) < rank(e_2)$, then according to Lemma 2.6 the tops of B_2 lie above $\text{top}(\pi)$, so π does not separate B_2.

Now consider the case[3] $rank(e_1) = rank(e_2)$. Let z be the z-coordinate corresponding to this rank (so h_z is the plane through the lower endpoints of e_1 and e_2). Since e' is a transfluent edge, the paths in B_2 and π cross the same triangle t' before encountering e'. We can assume that B_2 and π enter t' through the

[3]The argument for the case $rank(e_1) = rank(e_2)$ also applies when $e_1 = e_2$. This special case may happen when an up-path traverses some edges intersecting h_z in a cyclic way. It is then possible that some up-paths in $P_z(e_1)$ cross a sequence $\mathcal{S}'$ of edges before hitting e', while others first traverse a cycle of all edges intersecting h_z, before crossing $\mathcal{S}'$ and hitting e'.

same edge, as in Fig. 2.2(b), otherwise π surely cannot separate B_2. Now imagine following π backwards from e' as long as it follows the same edge-sequence as B_2. If π lies in between π_1 and π_2, the path π must follow the same edge sequence until either e_1 or e_2, whichever comes first. In fact, we can argue that e_2 must come first—otherwise, when π jumped to e_1 it would actually have stopped at e_2. We claim that π crosses e_2 above any of the paths $\pi_i \in B_2$, which then implies part (ii) of the lemma. Let t be the triangle that π and the paths in B_2 cross just before e_2 and let e'', e''' be the other two edges incident to t. Suppose π enters t through e'', as in Fig. 2.2(b) . There are two cases.

- First consider a path $\pi' \in B_2$ that also crosses e'' when it jumped to e_2. Let q be the point where π crosses e'', and let q' be the intersection point of π' and e''. Since $tail_\pi(q)$ crosses e_1 and $tail_{\pi'}(q')$ does not cross any edge with rank higher or equal to $rank(e_2)$ we have that $rank(tail_\pi(q)) > rank(tail_{\pi'}(q'))$. By Lemma 2.6 we get that q lies above q' on e''. Thus, by Lemma 2.5, the top of π on the forthcoming encounter with e_2 also lies above the top of π' on e_2 as claimed.
- Now consider a path π' that did not reach e_2 through e'', but through edge e'''. The lower vertex of e_2 is intersected by h_z, and e'' or a vertex of e'' is intersected by h_z since there is a path from e_1 that crosses e'', namely π, before hitting an exit feature. Then e''' must lie either completely below or completely above h_z, otherwise t is horizontal. Since π' crosses e''' before ever hitting e_2, then e''' can only lie below h_z. The fact that e''' lies below h_z and π crosses e'' above h_z implies that the top of π on e_2 lies above the top of π' on e_2, as claimed.

□

We now return to the algorithm, and show how it can be implemented efficiently.

The contour structure. Consider a situation where h_z does not contain a vertex. Then $h_z \cap \mathcal{T}$ consists of a number of simple, closed, polygonal curves, called *contours*. Let $C_1, C_2, \ldots$ be the contours, and let $\mathcal{S}_i$ denote the (cyclic) edge sequence corresponding to C_i. We give each edge $e \in \mathcal{S}_i$ that can be hit in clockwise direction by an up-path a label CW, and each edge that can be hit in counterclockwise direction a label CCW. Note that ridge edges get two labels, transfluent edges get one label, and valley edges get no label. We partition $\mathcal{S}_i$ into maximal subsequences $\mathcal{S}_i^j$ of edges with the same label; we call them CW-subsequences and CCW-subsequences depending on their common label. A ridge edge will be part of two subsequences (one CW-subsequence, and one CCW-subsequence), a transfluent edge will be part of one subsequence, and a valley edge will not be part of any subsequence.
Each subsequence $\mathcal{S}_i^j$ will be stored in an augmented tree $D(\mathcal{S}_i^j)$, which is the same as the cycle tree of the previous section, except for the following. First, the

trickle functions should be reversed, meaning that they should specify how an up-path (rather than a trickle path) can traverse a sequence. Second, each internal node $\nu \in D(\mathcal{S}_i^j)$ stores a boolean *unmarked*$[\nu]$ indicating whether any of the edges stored in the subtree rooted at ν is still unmarked. This way, when we jump over some edges of $\mathcal{S}_i^j$ to the first encountered edge above the sweep plane, we can mark all unmarked edges in logarithmic time per unmarked edge.
Inserting an edge or deleting an edge from the contour can be done in logarithmic time. Moreover, we can merge and split any of the structures $D(\mathcal{S}_i^j)$ in logarithmic time; this is necessary when we hit a saddle vertex, for instance, since then two contours split.

The bundle tree. Consider an edge e stored in the event queue with $P_z(e) = \pi_1, \dots, \pi_k$. Let $\text{tops}_z(e) = \tau_1, \dots, \tau_k$ be the tops of these up-paths on e. The bundle tree $T_{\text{bundle}}(e)$ stored with e is a balanced binary tree that we define as follows.

- The leaves of $T_{\text{bundle}}(e)$ store the tops $\tau_2, \dots, \tau_{k-1}$ in order. Let $\text{dist}(\tau_i, \tau_j)$ denote the distance between the tops τ_i and τ_j. A leaf node ν that stores the top τ_r also stores the ratio $\frac{\text{dist}(\tau_r, \tau_{r+1})}{\text{dist}(\tau_{r-1}, \tau_r)}$. This ratio remains the same when we expand the bundle upwards as long as the two paths incident to π_r follow the same sequence of edges.

- For an internal node ν, let $\text{first}[\nu]$ and $\text{last}[\nu]$ denote the tops stored in the leftmost and rightmost leaf below ν, respectively. Let $\text{pred}[\nu]$ be the top that comes before $\text{first}[\nu]$, and $\text{suc}[\nu]$ the top that follows $\text{last}[\nu]$. Then ν stores the ratios $r_1[\nu] = \frac{\text{dist}(\text{first}[\nu], \text{last}[\nu])}{\text{dist}(\text{pred}[\nu], \text{first}[\nu])}$ and $r_2[\nu] = \frac{\text{dist}(\text{last}[\nu], \text{suc}[\nu])}{\text{dist}(\text{pred}[\nu], \text{first}[\nu])}$.

- We store with $T_{\text{bundle}}(e)$ the coordinates of τ_1 and τ_k, and $\text{dist}(\tau_1, \tau_2)$ and $\text{dist}(\tau_{k-1}, \tau_k)$.

Updates such insertion and deletion on a bundle tree, but also merging and splitting, can be done in logarithmic time.
Next we show how to compute, given a point $p \in e$, which tops of $P_z(e)$ lie on each side of p. In other words, we have to determine the maximum j such that $\tau_j \in P_z(e)$ lies below p. We start by setting $\nu := \text{root}(T_{\text{bundle}}(e))$. We maintain the invariant that τ_j is stored in a leaf under ν, or $j = 1$, or $j = k$. Define $d := \text{dist}(\text{pred}[\nu], p)$ and $\delta := \text{dist}(\text{pred}[\nu], \text{first}[\nu])$. Initially we have $d = \text{dist}(\tau_1, p)$ and $\delta = \text{dist}(\tau_1, \tau_2)$. Also define $\delta_1 := \text{dist}(\text{pred}[\nu], \text{last}[lc[\nu]])$ and $\delta_2 := \text{dist}(\text{pred}[\nu], \text{first}[rc[\nu]])$. Note that $\delta_1 = \delta \cdot (1 + r_1[lc[\nu]])$ and $\delta_2 = \delta_1 + \delta \cdot r_2[lc[\nu]]$. Using the information stored in $T_{\text{bundle}}(e)$, we can maintain $d, \delta, \delta_1, \delta_2$ in constant time as we descend in $T_{\text{bundle}}(e)$. To determine to which child to proceed, we distinguish three cases:

(i) if $d < \delta_1$ then τ_j is stored in a leaf below $lc[\nu]$ or it is τ_1, and so we set $\nu := lc[\nu]$.

(ii) if $\delta_1 < d < \delta_2$ then τ_j is last$[lc[\nu]]$, and we are done.

(iii) if $\delta_2 < d$ then τ_j is stored in a leaf below $rc[\nu]$ or it is τ_k, and so we set $\nu := rc[\nu]$.

The process to find τ_j takes logarithmic time. After finding τ_j, we can split $T_{\text{bundle}}(e)$ in logarithmic time into a bundle tree T^1_{bundle} for $\pi_1, \ldots, \pi_j$ and a bundle tree T^2_{bundle} for $\pi_{j+1}, \ldots, \pi_k$.

Details of the algorithm. Now that we have described D_{contour} and T_{bundle}, we can explain steps 4, 9, and 14 of *ExpandUpNetwork* in more detail.

Step 4: Updating the contour structure. Whenever we move the sweep plane h_z upward to some new elevation z^*, we have to update D_{contour}: we must delete all edges whose top endpoint now lies on or below h_z, and we must insert all edges whose bottom endpoint lies on h_z. Updates can be done in $O(\log n)$, so in total they take $O(n \log n)$ time.

Step 9: expanding a vertex v. The number of up-paths emanating from v is at most the degree of v. Each up-path may require updating Q and then updating some set $P_z(e_\pi)$, which takes $O(\log n)$ time. Hence, the vertex expansions take $O(n \log n)$ time in total.

Step 14: jumping from an edge e. To split $P_z(e)$ into bundles and jump with each bundle to its exit edge, we proceed as follows. Let $P_z(e) = \pi_1, \ldots, \pi_k$, let $\tau_1, \ldots, \tau_k \in e$ be the tops of these up-paths, and let $\mathcal{S}_i^j$ denote the subsequence (in the current set of contours) containing e. The call *FindExit*$(D(\mathcal{S}_i^j), q)$ reports, given a point q on an edge e intersecting the sweep plane h_z, the first feature f_{exit} crossed by q's up-path that lies completely above h_z.
We first perform a query *FindExit*$(D(\mathcal{S}_i^j), \tau_1)$, giving us the exit feature $f_{\text{exit}}(\pi_1)$. Let $F_1 : e \to f_{\text{exit}}(\pi_1)$ be the function that maps a point $q \in \text{Dom}(F_1)$ to the point on f_{exit} that we reach when we follow an up-path from q. We modify *FindExit*$(D(\mathcal{S}_i^j), q)$ such that it also returns F_1 and $\text{Dom}(F_1)$. Let $T_{\text{bundle}}(e)$ be the tree storing $P_z(e)$. Using $T_{\text{bundle}}(e)$ we determine the largest j such that $\tau_j \in \text{Dom}(F_1)$ and we split $T_{\text{bundle}}(e)$ into two bundle trees T^1_{bundle} and T^2_{bundle}, as described above. By Lemma 2.7(i) the paths $\pi_1, \ldots, \pi_j$ follow the same edge sequence from e to $f_{\text{exit}}(\pi_1)$, thus forming the first bundle of $P_z(e)$.
We repeat the process with the remainder of $P_z(e)$, now stored in T^2_{bundle}, until we have determined all the bundles, and for each bundle b its exit feature $f_{\text{exit}}(b)$. For each bundle we then mark all newly crossed egdes—this will take $O(\log n)$ per marked edge—and if $f_{\text{exit}}(b)$ is an edge we insert b into $P_z(f_{\text{exit}}(b))$. The latter operation takes $O(\log n)$, since by Lemma 2.7(ii) b does not interleave with the up-paths already stored in $P_z(f_{\text{exit}}(b))$, which means we can add T^1_{bundle} to $T_{\text{bundle}}(f_{\text{exit}})$ by one splitting and two merging operations. In the case that b hits a ridge edge, we discard b and insert the upper vertex of this edge in Q.

Theorem 2.8 *Algorithm ExpandUpNetwork*$(\mathcal{T}, p)$ *computes for a point* p *on a terrain* $\mathcal{T}$ *with* n *vertices the points where the up-network from* p *ends, and the edges crossed by this up-network in* $O(n \log n)$ *time.*

Proof. To prove the time bound, it suffices to argue that $O(n)$ bundles are generated. When handling an edge e, a bundle is split off when the paths of $P_z(e)$ enter a triangle t through one edge e_1, but leave t through different edges e_2 and e_3. Let v be the common vertex of e_2 and e_3. According to Lemma 2.7 the up-paths of any other set $P_{z'}(e')$ do not interleave with $P_z(e)$ on e_1, and thus only one multi-bundle can split around v. □

2.4 Extracting Other Drainage Information

As described in Chapter 1, the surface network graph of a terrain $\mathcal{T}$ is an abstraction of the topology of $\mathcal{T}$. Recall that the vertices of this graph represent the critical points on the surface of $\mathcal{T}$, and the edges of the graph represent paths which follow the steepest slope on the surface of $\mathcal{T}$ and which connect saddles with local minima and local maxima. These paths are defined as follows. For a saddle vertex $v \in \mathcal{T}$ consider the region of $\mathcal{T}$ that consists of all the triangles incident to v. We call this region the *star* of v and we denote this by *star*(v). We call the *upper star* of v the subregion of *star*(v) that consists of the points that have a higher elevation than v. Similarly, we call the *lower star* of v the set of points in *star*(v) that have a lower elevation than v. As v is a saddle vertex, the upper star and the lower star of v consist of at least two connected components each. For each connected component of the upper star (lower star) of v we consider a unique ascending (descending) path: this is the path that stems from v towards the direction of steepest slope within this component.
To construct the surface network graph of $\mathcal{T}$ we need to compute for each saddle vertex v the set of local extrema where the desrcibed paths from v end. For this we need the following more general versions of the algorithm *ExpandUpNetwork*. Let P_{saddle} be the set of the $O(n)$ saddle points on $\mathcal{T}$. Then we can compute the edge-set of the surface network graph in $O(n \log n)$ time; first we initialise the event queue Q in Step 1 of *ExpandUpNetwork* with the points of P_{saddle}. At this step we expand only the up-paths from the saddle points. It is easy to see that the algorithm *ExpandUpNetwork* has the same running time when expanding $O(n)$ paths simultaneously on $\mathcal{T}$ even if these paths do not emanate from the same point. When we initiate an up-path π, we associate π with the critical point $v[\pi]$ from which the path emanates. An up-path is terminated when it hits a terrain feature that is a vertex or a ridge edge. If this feature is a critical point u we add an edge $(v[\pi], u)$ in the surface network graph, otherwise we propagate the tag $v[\pi]$ to the path of steepest ascent that starts from this feature. To compute the rest of the edges of the graph we use an algorithm *ExpandDownNetwork*, which is essentially the same as *ExpandUpNetwork* except that it traces paths downwards

instead of upwards. In the proof of the following theorem we also show how we can compute the triangles contained in the watershed of each local minimum on $\mathcal{T}$ in $O(n \log n)$ time.

Theorem 2.9 *Let $\mathcal{T}$ be a terrain with n triangles and let P be the set of local minima on $\mathcal{T}$. We can compute the surface network graph of $\mathcal{T}$, and assign to each minimum $p \in P$ the triangles that are entirely contained in the watershed of p in $O(n \log n)$ time.*

Proof. Consider a local minimum p of $\mathcal{T}$ and let t be a triangle that is entirely contained in the watershed of p. That means that the trickle path from every point in the interior of t ends in p. This can only happen if these trickle paths (except maybe a discrete subset of these paths) contain also one or more valley edges. Hence, in order to compute the watershed of p we have to find which valley edges send water to p and then find the triangles that send water to these edges. Therefore we proceed as follows.
We use *ExpandDownNetwork* to compute simultaneously for each down-path of each terrain vertex v the first valley edge hit by such a path; the algorithm can also compute exactly the points where these paths hit their first valley edge. Now consider a valley edge e whose lowest endpoint sheds water to a local minimum p, and suppose e is the first valley edge hit by the down-paths of vertices $v_1, \ldots, v_k$. Let $q_i \in e$ be the point where *trickle*(v_i) hits e, and assume $z(q_1) < z(q_2) < \ldots z(q_k)$. For ease of exposition, we assume that these points $q_1, \ldots q_k$ are all different—the reader may verify that the following proof could easily be adapted to accommodate situations in which a trickle path reaching e from the left and a trickle path reaching e from the right reach e in exactly the same point. Define q_0 and q_{k+1} to be the lowest and highest endpoints of e, respectively. The points q_i for $0 \leqslant i \leqslant k$ are the lowest vertices of the *strips* [96] incident to the edge e. A strip is a maximal subset of the terrain surface extending between a segment s of a valley edge and a segment of a ridge edge such that all up-paths starting from s traverse the same sequence of edges. For $0 \leqslant i \leqslant k$, let $p_i \in e$ be a point that we pick arbitrarily between q_i and q_{i+1}. Imagine tracing an up-path from each p_i, leaving in the direction where *trickle*(v_i) comes from, until a ridge edge is reached. It can be shown [96] that the triangles containing a point q for which e is the first valley edge hit by *trickle*(q), are precisely the triangles crossed by one of these up-paths. We collect the points p_i, q_i over all valley edges in a set Q, and then apply a modified version of *ExpandUpNetworkTriangle* to Q. In this version of the algorithm we associate each terrain edge e with a tag that indicates if all the trickle paths starting from points on e end at the same local minimum or not. Next we describe shortly what is the reason for using these tags and then we continue with the description of the algorithm.
Let e be a valley edge and let v the lowest vertex incident to e. We tag e with the local minimum where *trickle*(e) ends. We tag each up-path in Q with the same local minimum as the valley edge where it comes from. A triangle t is contained in the watershed of a local minimum p if and only if the valley and transfluent edges

of t are intersected only by up-paths in Q that are tagged with p. If the valley and transfluent edges of t are intersected by up-paths that have different tags then t is a border triangle.
For each bundle tree T_{bundle} that is generated during the sweep we maintain a tag $tag[T_{\text{bundle}}]$ in the following manner: if a bundle tree T_{bundle} stores up-paths that are all tagged with the same local minimum p then we have $tag[T_{\text{bundle}}]$ = "p" otherwise this tag has a symbolic value "MULTIPLE". We store also such a tag for every node of T_{bundle}, maintaining this information for each subtree of T_{bundle}. In this way, whenever a new bundle tree T'_{bundle} is generated from splitting or merging other bundle trees then the value of $tag[T'_{\text{bundle}}]$ can be computed in $O(\log n)$ time.
We also change the fields stored with each node ν of a tree $D(\mathcal{S}_i^j) \in D_{\text{contour}}$ slightly. Instead of a boolean *unmarked*$[\nu]$, we store a tag $tag[\nu]$. If ν is a leaf node, then ν represents an edge crossed by h_z. Let $e[\nu]$ be this edge. The value stored in $tag[\nu]$ may be of three possible kinds:

- If $e[\nu]$ has not been crossed so far by any up-path then $tag[\nu]$ stores a symbolic value "NONE".
- If $e[\nu]$ has been crossed only by up-paths that were tagged with the same local minimum p then $tag[\nu]$ ="p".
- If $e[\nu]$ has been crossed by up-paths that were tagged with different local minima then $tag[\nu]$ ="MULTIPLE".

For an internal node $\nu \in D(\mathcal{S}_i^j)$ let T_ν be the subtree of $D(\mathcal{S}_i^j)$ with root ν. If all the leaves in T_ν have the same tag value then $tag[\nu]$ is also set to this value. Otherwise, we distinguish two more cases. If the only tags that appear in the leaves of T_ν are "MULTIPLE" and "p" for only one local minimum p, then $tag[\nu]$ ="MULTIPLE AND p". In any other case $tag[\nu]$ ="MIXED". Notice that $tag[\nu]$="MULTIPLE" means that each valley edge represented by a leaf node in T_ν has been crossed by up-paths that were tagged with different local minima. However, $tag[\nu]$="MIXED" implies that there are two or more leaf nodes in T_ν that have different tags with each other; for example there may exist a leaf node ν' with $tag[\nu']$="p" and a leaf node ν'' with $tag[\nu'']$="q" because $e[\nu']$ was crossed only by up-paths tagged with "p" while $e[\nu'']$ was crossed only by up-paths tagged with "q".
Suppose that we execute a query *FindExit*$(D(\mathcal{S}_i^j), \tau)$ for some up-path τ and for some tree $D(\mathcal{S}_i^j) \in D_{\text{contour}}$ that stores a CW or CCW subsequence. Let T_{bundle} be the bundle tree that is generated after this query and which stores τ. Let $\nu \in D(\mathcal{S}_i^j)$ be a node encountered during this query such that τ was found to traverse all the edges stored in the subtree with root ν. We distinguish the following cases:

- ◦ If $tag[\nu]$="NONE" then we assign $tag[T_{\text{bundle}}]$ to $tag[\nu]$ and we do the same for all the nodes in T_ν.
- ◦ If $tag[\nu]$ = "MULTIPLE" then we do not change anything.

- If $tag[\nu]$ corresponds to a local minimum p then we check the tag of T_{bundle}; If also $tag[T_{\text{bundle}}]$ =“p” then we do not change anything, otherwise we set $tag[\nu]$ =“MULTIPLE” for all the nodes in T_ν.
- If $tag[\nu]$ =“MULTIPLE AND p” then if $tag[T_{\text{bundle}}]$ =“p” we do not change anything, otherwise we set $tag[\nu]$ =“MULTIPLE” and we recurse with the children of ν.
- If $tag[\nu]$ =“MIXED” then if $tag[T_{\text{bundle}}]$ =“MULTIPLE” we set to “MULTIPLE” the tag for all the nodes in the subtree with root ν. Otherwise, if $tag[T_{\text{bundle}}]$ =“p” for some local minimum p we recurse with the children of ν.

According to the above, changing the values of the $tag[\cdot]$ fields of the nodes takes in total $O(\log n)$ steps for each leaf node that was updated. This is because we only visit the children of any node ν if at least one leaf tag in the subtree rooted at ν will be changed. We charge the cost of visiting nodes in the tree to these leafs, thus each leaf whose tag changes is charged by at most $O(\log n)$ children of its ancestors. The tag of each leaf node in the contour structure will be updated at most twice during the execution of the algorithm which takes $O(n \log n)$ time in total.
After executing the modified version of *ExpandUpNetwork* we check for each terrain triangle the tags of its incident edges and assign this triangle to a watershed of a local minimum or classify it as a border triangle accordingly.

□

We can use a variant of *ExpandDownNetwork* to compute the exact watershed area for each local minimum on $\mathcal{T}$ in $O(n^2)$ as explained in the following theorem.

Theorem 2.10 *Let $\mathcal{T}$ be a terrain with n triangles and let P be the set of local minima on $\mathcal{T}$. The exact measure of the area covered by the watershed of each point $p \in P$ can be computed in $O(n^2)$ time.*

Proof. Let p, q be two points on the interior of an edge $e_1 \in \mathcal{T}$ and let π_p and π_q be the up-paths that start from these points respectively. Suppose that these two up-paths cross a common sequence of edges $\mathcal{S} = e_1 e_2 \dots e_k$ and suppose no edge occurs in $\mathcal{S}$ more than once. Let p' and q' be the intersection points of π_p and π_q, respectively, with e_k. Let Λ be the part of $\mathcal{T}$ that is bounded by $\overline{pq}$, $\overline{p'q'}$, π_p, and π_q. The area of Λ can be expressed as a quadratic function $K_{\mathcal{S}}$ of the coordinates of p and q. We call this function the *area function* of $\mathcal{S}$. It is important to note that the value of $K_{\mathcal{S}}$ does not depend only on the length of $\overline{pq}$ but also on the exact position of p, q.
To compute the area of the watershed of each local minimum in P we proceed as follows. We use *ExpandDownNetwork* to compute for each valley edge e the intersection points of e with the paths of locally steepest descent that start from

vertices of $\mathcal{T}$. Let $q_1(e), q_2(e), \ldots, q_k(e)$ be the intersections points of e with these paths. Assume $z(q_1(e)) < z(q_2(e)) < \ldots < z(q_k(e))$, and let $q_0(e)$ and $q_{k+1}(e)$ to be the lowest and highest endpoints of e respectively. The segments $\overline{q_i q_{i+1}}$ for every $0 \leqslant i \leqslant k$ bound from below the strips—see Section 1.1.3 for a definition—that are incident to e. As shown by Yu *et al.* [96], each strip is a region entirely contained in the watershed of some local minimum. Our approach will be to compute the area of each of the strips simultaneously and then sum the computed values of the strips that are associated with the same local minimum. For $0 \leqslant i \leqslant k$, let $p_i(e)$ be a point that we pick in an arbitrary way on the interior of $\overline{q_i(e) q_{i+1}(e)}$.
For each valley edge $e \in \mathcal{T}$ we insert all the points $p_i(e)$ that we constructed into an initially empty queue Q. We maintain for each $p_i(e)$ a quadratic function $K[p_i(e)]$ that is initially set to zero, and we apply a new version of *ExpandUpNetwork* to Q.
For this version of the algorithm we store two extra quadratic functions with each node ν of a tree $D(\mathcal{S}_i^j) \in D_{\text{contour}}$ that stores a CW/CCW subsequence. In detail, node ν stores the quadratic function $K_{\mathcal{S}[\nu]}$ and the quadratic function $K_{\mathcal{S}'[\nu]}$ with $\mathcal{S}[\nu]$ and $\mathcal{S}'[\nu]$ defined as in Section 2.2. The following formula shows how we can compute $K_{\mathcal{S}[\nu]}$ in constant time given the satellite data of the children of ν:

$$K_{\mathcal{S}[\nu]} = K_{\mathcal{S}[lc[\nu]]} + K_{\mathcal{S}'[\nu]} \circ F_{\mathcal{S}[lc[\nu]]} + K_{\mathcal{S}[rc[\nu]]} \circ F_{\mathcal{S}'[\nu]} \circ F_{\mathcal{S}[lc[\nu]]}$$

Consider a call *FindExit*$(D(\mathcal{S}_i^j), \tau)$ for some tree $D(\mathcal{S}_i^j) \in D_{\text{contour}}$ that stores a CW/CCW subsequence, and some up-path τ. Let $\mathcal{S}$ be the sequence of edges that τ traversed during this call. In this new version of *FindExit* we compute also the area function $K_{\mathcal{S}}$ as a sum of quadratic functions stored with at most $O(\log n)$ nodes in $D(\mathcal{S}_i^j)$. Let T_{bundle} be the bundle that contains τ. At the the end of the call of *FindExit* we add $K_{\mathcal{S}}$ to $K[p_i(e)]$ for every $p_i(e)$ which is the starting point of an up-path in T_{bundle}. This takes $O(n)$ time for each generated bundle instead of $O(\log n)$ which was the case for the basic version of *FindExit*. Thus the overall running time of *ExpandUpNetwork* becomes $O(n^2)$.
After the execution of *ExpandUpNetwork* we associate with each local minimum $p \in P$ a watershed area value $A[p]$ initially set to zero. For each edge e, we apply each function $K[p_i(e)]$ to the points $q_i(e), q_{i+1}(e)$ and then add the computed value to $A[p]$, where p is the local minimum at which *trickle*$(q_i(e))$ and *trickle*$(q_{i+1}(e))$ end. The resulting value $A[p]$ is the exact watershed area of each local minimum $p \in \mathcal{T}$. □

2.5 Concluding Remarks

In this chapter, we presented efficient algorithms that compute certain flow-related structures on terrains and their characteristics: the surface network, an approximation of the watersheds of all local minima, and the exact area for the watersheds

of all local minima on the terrain. Our algorithms have an improved worst-case running time compared to previous approaches that involve computing paths of steepest ascent/descent on the terrain explicitly. An interesting problem for future research is to design an implicit mechanism that can answer questions related to proximity structures on terrains such as Voronoi diagrams that are defined according to the geodesic distance on the terrain surface.

Chapter 3

Computing Drainage Structures on TINs: Practical Issues

3.1 Introduction

As indicated in the previous chapters, triangulated terrain models provide a strong advantage when it comes to flow modelling; they are continuous surfaces. Thus, we can consider an intuitive geometric interpretation for modelling the course of water on such surfaces: we assume that water follows always the direction of steepest descent on the terrain (DSD) and this direction is unique for every point on this surface. We termed this model as the DSD model.
Given a point p on a terrain $\mathcal{T}$, a simple task which is important for many hydrological applications is to compute the *trickle path* of p. This is the path that starts from p and follows the DSD on the surface of $\mathcal{T}$ until it reaches a local minimum or the boundary of $\mathcal{T}$. However, even the task of computing trickle paths on TINs is not as easy as it seems. First of all, the DSD model does not specify how water flows across flat (horizontal) areas or when the DSD is not unique. How to deal with this is an important research topic in itself, which is complementary to the topic that we address in this chapter. Second, even when the DSD is unique everywhere, it is not easy to compute trickle paths in an exact and robust manner. The difficulty lies in the fact that the trickle path does not necessarily follow edges of the TIN—it sometimes crosses triangle interiors. This can cause robustness problems during the computations: the use of standard, fixed-precision arithmetic may lead to incorrect results. Note that a small error upstream in a flow path may cause a very large deviation downstream.
Another important algorithmic question—the one that is the main focus of this

chapter—is how to compute watersheds. Recall that the *watershed* of a point p on $\mathcal{T}$ is the region consisting of all points whose trickle path contains p, and the *watershed map* of $\mathcal{T}$ is the subdivision of $\mathcal{T}$ induced by the watersheds of all local minima. Again, the fact that water can flow through triangle interiors makes the exact computation of watersheds difficult, both conceptually and from an implementation point of view. The use of finite-precision computations can even lead to inconsistent results such as a watershed containing no, or more than one, local minimum [57] and, if one is not very careful, the program crashes.

Previous work There exist many algorithms that compute a watershed subdivision on a given TIN; an overview can be found in the paper by Čomić *et al.* [24]. Due to problems mentioned above, most of these algorithms do not follow the DSD model exactly. Instead, they often only consider flow along edges of the TIN. Thus, they restrict the flow to a discrete network rather than considering the whole TIN surface. One example of this approach is the popular algorithm of Mangan and Whitaker [60]; other examples are the methods of Takahashi *et al.* [87] and of Vincent and Soile [92].

Using only the TIN edges to propagate flow affects, of course, the quality of the output. One of the consequences is that the computed watersheds are collections of full triangles. However, according to the DSD model it is possible that watershed boundaries extend through triangle interiors. (Instead of considering the TIN edges, some methods [68] consider a network that also includes selected points on triangle interiors, leading to similar problems.) Because local errors in the flow can have global effects, they can even assign triangles to the wrong watershed when no watershed boundary crosses them. From now on we refer to flow models (or algorithms) that do not strictly follow the DSD assumption as *inexact flow models* (or algorithms). The DSD model will sometimes be referred to as the *exact flow model*.

As mentioned in the introduction of this thesis, de Berg *et al.* [6] were the first to study the complexity of various structures in the DSD model on a TIN. For example, they showed that in the worst case a single trickle path may cross the same TIN edge several times. In fact, in a worst-case (and very unrealistic) scenario, a single trickle path on a TIN of n triangles can cross many edges many times, leading to a worst-case complexity of $\Theta(n^2)$. They also studied the worst-case complexity of watersheds and *river networks*—see Section 1.1.2 for a precise definition.

The DSD model was further investigated by Yu *et al.* [96]. They introduced a key concept for modelling drainage areas, the so-called *strip*. As explained also in Chapter 1, if we expand from each TIN vertex all paths of locally steepest ascent and descent we get a refinement of the terrain surface which is called the *strip map*. The expanded paths, the ridge edges and the valley edges of the TIN subdivide the terrain into faces, that is, the strips. Recall that a valley edge is an edge such that the DSD in the interior of both of its incident triangles points towards this edge, while for a ridge edge the DSD in the interior of both of its incident triangles points

away from this edge. Each strip is bounded from the top by a ridge edge segment and from the bottom by a segment of a valley edge. Its side boundaries consist of two or more paths of steepest ascent/descent—see Fig. 1.4 for a depiction of the topology of several strips and Fig. 3.1 for a global view of the strip map of a TIN. If we expand the trickle path of any point in the interior of a strip then this path will hit the valley edge segment at the strip bottom and thus it will end at the local minimum where this valley edge drains. Thus the watershed of a local minimum in the watershed map is the union of one or more strip interiors and boundaries, and the watershed map is easily extracted from the strip map. Unfortunately, as mentioned in the introduction of this thesis, the complexity of the strip map is quite high. McAllister [2], and Jones *et al.* [47] presented algorithms for computing the watershed map on a TIN without constructing the complete strip map; instead both of these approaches only consider paths of steepest descent/ascent from a specific subset of the TIN vertices. Unfortunately, this may lead to incomplete results [7].
When it comes to up-to-date implementations of watershed algorithms according to the DSD model, we are only aware of implementations of McAllister's algorithm [57, 2, 3]. The most detailed discussion of their implementation is given by Liu and Snoeyink [57]. As discussed in Chapter 1, they also consider, from a theoretical point of view, numerical issues that arise when implementing flow computations according to the DSD model. More specifically, they examine how many bits are needed to represent the exact coordinates of the intersection points between a trickle path and the TIN edges crossed by this path. They conclude that this number grows as a linear function of the number of transfluent edges crossed by the path. However, in their implementation they make use of fixed-precision arithmetic which may lead to inconsistencies in the output. This leads to the following questions: Is it feasible in practice to compute trickle paths and watersheds on a TIN *exactly* according to the DSD model, using an algorithm based on the complete strip map and using exact arithmetic? What are the bit-sizes needed in the computations? And if the exact algorithm turns out to be impractical, then which of the inexact methods approximates the exact results best?

Our contribution We provide the first complete and exact implementation that computes watersheds on TINs according to the DSD model. Our implementation is based on the computational-geometry algorithms library CGAL [21], which provides an easy way to perform the computations using exact arithmetic. We experimentally investigate the performance of our implementation on several real-world data sets. In particular we measure the bit-sizes needed for the exact computations and the resulting memory usage of the algorithm. As it turns out, the large bit-sizes are not only a problem in theory, but also in practice. Moreover, the computation of the complete strip map is a significant bottleneck of the algorithm: the strip map has a much higher complexity than the final watershed map computed from it (and this problem is aggravated by the large bit-size problem). Hence, our implementation is impractical for large data sets.

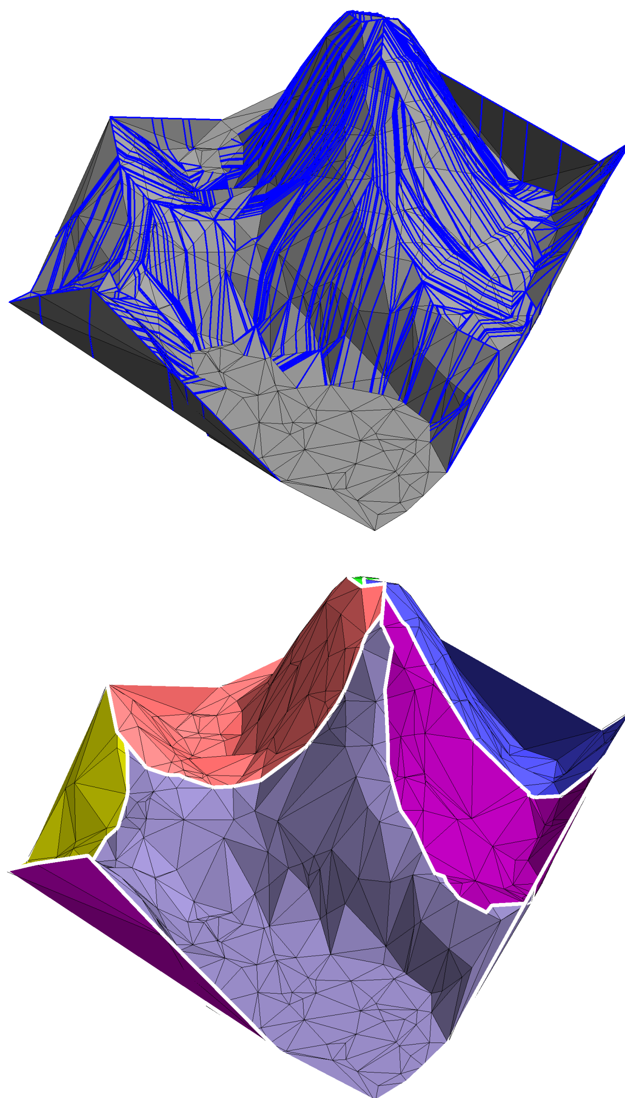


Figure 3.1 Drainage structures on a TIN: (top) The strip map. Strip boundaries are highlighted in blue colour. Note that strips do not extend through the flat regions of the terrain. (bottom) The watershed map. Watershed boundaries are highlighted in white colour.

Nevertheless, our exact algorithm provides us with the opportunity to study the quality of the output of inexact (but hopefully more efficient) algorithms, since it

can serve as a point of reference. Therefore we implement three different inexact algorithms from the literature—the algorithm by Mangan and Whitaker [60], the one by Takahashi *et al.* [87], and the one by Vincent and Soile [92]—and we compare their output to the output of our exact algorithm. All algorithms are quite efficient, but the algorithm by Mangan and Whitaker [60] turns out to produce the highest-quality results. We also propose and investigate hybrid methods, which are based on the above-mentioned heuristics, but perform part of the computation in an exact manner. These hybrid approaches turn out to be almost as fast as the heuristics, while giving substantially more accurate results.

As mentioned above, the explicit computation of the complete strip map is a major bottleneck. In Chapter 2 we showed how to compute various flow-related structures in an implicit manner, thus greatly speeding up the worst-case theoretical running times. Based on ideas from that chapter we describe an algorithm to compute the exact watershed map without computing the complete strip map as an intermediate structure. Recall that in Chapter 2 we described an algorithm that produces only an approximation of the watershed map of a terrain, assigning to each watershed w only those triangles that are fully contained in w. We perform a theoretical analysis of the running time of our algorithm as a function of the input size n and the output size k (that is, the size of the watershed map). The analysis shows that if the watershed map has small complexity—which is the case in practice, as our experiments have shown—then its theoretical running time is superior to the running time of algorithms that compute the complete strip map.

3.2 Description of Implementation and Experiment Settings

In this section we describe our exact implementation for computing watersheds according to the DSD model, and the set-up for our experiments. Our software provides algorithms for computing flow paths, watersheds, strip maps, river networks and surface networks on TINs.

3.2.1 Implementation Using CGAL

As mentioned in Section 3.1, Liu and Snoeyink [57] showed that fixed-precision arithmetic is insufficient for representing the coordinates of the intersection points between flow paths and terrain edges exactly. Indeed, as we follow a trickle path, it may cross a sequence of distinct transfluent edges $\{e_1, e_2, \ldots, e_k\}$—recall that transfluent edges are edges that are neither valley edges nor ridge edges—and each crossing can increase the bit-size of the intersection point. More precisely, if the coordinates of the starting point of the trickle path and the coordinates of each vertex of $\mathcal{T}$ are numerical values of constant bit-size, then a single coordinate of the intersection point between this path and edge e_i can be a rational with bit-size $\Theta(i)$. Hence, using doubles or any other fixed-precision representation is insufficient to represent these coordinates exactly. The resulting inconsistencies may cause severe problems. Indeed, running our implementation using fixed-precision arithmetic (e.g. doubles) often results in program crashes. We thus need *exact arithmetic*.
We therefore base our algorithm on the *Computational Geometry Algorithms Library* (CGAL). CGAL is an open source software library in C++ that provides a wide range of geometric algorithms and data structures. This includes both basic subroutines (computing intersection points of geometric objects, distances, etcetera) as well as more involved algorithms and data structures (for convex hulls, Voronoi diagrams, point location, and so on). Two major advantages of CGAL are the flexibility that comes from the use of generic programming, and computational robustness through the use of exact arithmetic.
CGAL follows the generic-programming paradigm, making heavy use of templates that provide the opportunity to define a class using different parameter-types for its representation. For example, an important notion in CGAL are the *geometric kernels*: classes that provide the definitions of essential geometric objects and functions applied to these objects. A kernel is a template class that takes as a parameter the number type that represents the encapsulated geometric objects and functions. Hence, by instantiating the Cartesian kernel of CGAL using the `double` C++ built-in type as a template parameter, the encapsulated object-types of the kernel such as points or segments are represented with cartesian coordinates of double floating point precision. It is the user then who chooses which number type to use and makes the trade-off between exact (but sometimes slow) and fast (but not exact) computations.
In our implementation we use the GNU number type `Gmpq` [38], which is appropriate for computations between rationals of arbitrary precision. In fact, we use the more refined CGAL number type `CGAL::Lazy_exact_nt<Gmpq>` [45]. The latter number type uses an *algebraic filtering technique*: it uses fixed-precision arithmetic whenever it is possible to avoid costly exact computations, but it switches to exact arithmetic when fixed precision is not enough to determine the output of some predicate correctly. This filtering technique makes it possible to construct topologically correct flow paths (and derived drainage structures) with smaller bit-size

than the bit-size of their exact representation. At the end of the computation, it is still possible to extract the exact coordinates. The watersheds in the output are represented as graph type objects that follow the standards of the Boost Graph Library [34].
The algorithm we used for computing the watershed map of an input TIN $\mathcal{T}$ is inspired by the work of Yu *et al.* [96]. First we compute the strip map of $\mathcal{T}$ by expanding from each vertex $v \in \mathcal{T}$ the paths of locally steepest descent and the paths of locally steepest ascent. In the resulting subdivision, all points in the interior of the same strip have their trickle paths overlapping on the lower strip boundary and thus they drain to the same local minimum. Hence, we label the subfaces of triangles within each strip with the local minimum to which the strip drains. Then we delete from the strip map the parts of the constructed paths that are adjacent to strips labelled with the same local minimum [1]. The resulting subdivision is then the watershed map of $\mathcal{T}$.

3.2.2 Experimental Set-Up

The experiments that are presented in this chapter were conducted using an Intel i5 four-core CPU where each core is a 3.20 GHz processor. However, as there is no parallelization in the algorithms that we implemented, the computations were handled each time by only one of the processors. The main memory of this system is 3.4 Gigabytes. Our code runs on a Linux Ubuntu operating system version 10.10 using the GNU g++ version compiler 4.4.4. Our implementation is compatible with version 3.8 of CGAL.
The TINs we have used in our experiments were constructed using data obtained from the U.S. Geological Survey (USGS) online server [90]. Each data set is a DEM in the `ADF` file format and consists of a regular 3,612 × 3,612 grid at 30m resolution, where for each grid cell the elevation is given as a 4-byte floating point value. Each data set represents a certain region in the United States. The names that we use to refer to each of these data sets, as well as their elevation ranges and the geographical areas that they model, are summarized in Table 3.1.

Table 3.1 The data sets used in the experiments.

name	modelled region	elevation range
`duchesne`	Duchesne (UT)	[1412 m, 3737 m]
`nazareth`	Nazareth (TX)	[1051 m, 1349 m]
`parnassus`	Mount Parnassus (CO)	[1551 m, 4351 m]

To construct each input TIN, we selected 1,201 × 1,201 grid cells from the central region of the DEM. We chose to model this restricted region of the DEM to reduce

[1] If a (part of a) terrain edge appears on the boundary of adjacent strips that drain to the same minimum we just do not highlight this (part of the) edge as a watershed boundary

undersampling effects (as compared to modelling the whole region), while still modelling a reasonably "interesting" region. From this region we then sampled n points (centers of grid cells) that constitute the vertex set of the TIN, using a greedy method that attempts to minimize the elevation difference between the TIN surface and the remaining points [65]. The value of n depends on the experiment. In case neighbouring vertices have the same elevation, we apply a small perturbation, so that the flow is always well defined and the issue of how to deal with flat areas does not influence our results.

3.3 The Complexity of Flow Structures

In this section we present our experimental investigation of the complexity of several flow structures on TINs. Our goal is to provide an evaluation of both the combinatorial complexity of TIN drainage structures and their total bit-size complexity when the computations are done exactly.

3.3.1 The Complexity of Flow Paths

In the first set of experiments we measure the combinatorial complexity and the bit-size of individual flow paths that are expanded from the vertices of a TIN. The results of such experiments can provide insight on how the complexity of more complicated drainage structures grows as the input size increases. Indeed, the strip map of a TIN, and therefore also its watershed map, are computed by expanding paths of steepest gradient from every vertex of the TIN. If the average bit-size of individual flow paths rises considerably as the size of the input increases, then this poses a significant restriction on the size of the TINs for which we can compute the strip map and the watershed map using exact arithmetic. Next we provide a detailed description of the first set of our experiments.

Consider a vertex v of a TIN $\mathcal{T}$. A path of steepest descent or steepest ascent on $\mathcal{T}$ consists of line segments, which either lie in the interior of a TIN triangle or are (a part of) a TIN edge. The number of line segments the path consists of is called its *combinatorial complexity*. The total number of bytes needed to represent the coordinates of the vertices of the path—note that these are not necessarily TIN vertices—is called the *bit-size* of the path. For each vertex v of $\mathcal{T}$ we construct two paths, namely the path of steepest descent and the path of steepest ascent from v. We measure the following quantities as a function of n, the number of TIN vertices.

$cc(n)$: the average combinatorial complexity of the paths;
$bs(n)$: the average bit-size of the paths.

Note that the paths that we consider can overlap significantly. For example, when the path of steepest descent from a vertex v passes through a vertex w (usually after first following a valley edge) then from that point on it will coincide with the steepest-decent path from w. We therefore also measure the average *exclusive combinatorial complexity* and the average *exclusive bit-size* of the paths, denoted by $cc^*(n)$ and $bs^*(n)$, respectively. These are defined in the same way as $cc(n)$

and $bs(n)$, except that we only consider the part of each path up to the first encountered TIN vertex (after the starting point). Studying the complexity of this part of the path alone is important for estimating the size of the strip map; trickle paths and up-paths of vertices may overlap to a large extent and if we would sum the total complexity of each path individually we would overestimate the size of the entire structure.
We have computed the described complexity values for flow paths on TINs of different values of n but constructed from the same DEM data set. More specifically, from the `nazareth` DEM data set we have constructed 50 TIN instances, with $n = 1000k$ and k ranging from 1 to 50. The results of these experiments are presented in Figs. 3.2 and 3.3.

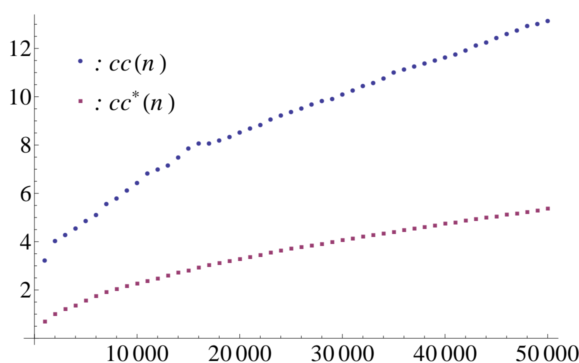


Figure 3.2 The average combinatorial complexity $cc(n)$ and average exclusive combinatorial complexity $cc^*(n)$ of the paths of steepest ascent and descent on the `nazareth` data set, as a function of the number of TIN vertices.

Discussion As can be seen in Fig. 3.2, the total (exclusive) combinatorial complexity of the paths increases with n (not surprisingly), though not linearly. Moreover, the average exclusive complexity is rather small: most paths quickly merge with other paths. The maximum combinatorial complexity of a single path (not shown in the figure) for the terrain with 50,000 vertices was 66. The average bit-sizes grow faster than the combinatorial complexity of the paths, indicating that the bit-sizes of the individual vertices on the paths are increasing. For a TIN of 50,000 vertices, the average exclusive bit-size is close to 1.3 kilobytes while the average exclusive combinatorial complexity is roughly 5. Since a path with five segments has six vertices, each having an x-, a y-, and a z-coordinate, this means we need about 72 bytes per coordinate. The maximum number of bytes for a single coordinate (not shown in the figure) was even 708 bytes, showing that very large bit sizes really arise in practice when doing exact computations. To represent all the paths together, we need approximately 130 MB, which is more than 100 times

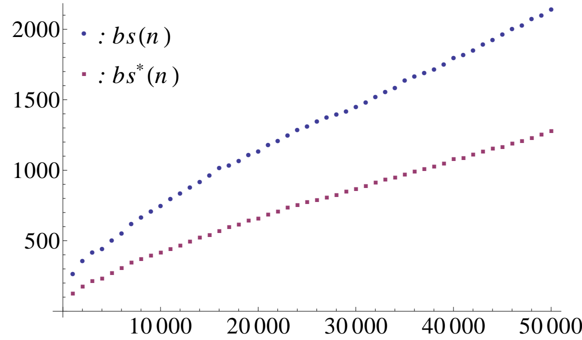


Figure 3.3 The average bit-size $bs(n)$ and average exclusive bit-size $bs^*(n)$ of the paths of steepest ascent and descent on the `nazareth` data set, as a function of the number of TIN vertices.

the 1.2 MB that would be needed for the TIN vertices if we used double precision floating point arithmetic. Since we have $2n$ paths, and the average exclusive complexity is 5, a factor 10 of the blow-up is caused by increasing combinatorial complexity. Another factor 10 is caused by the increase in the bit-size of the coordinates, which is thus a serious problem in practice. We will refer to these results later on, when we will experimentally evaluate the bit-size and the combinatorial complexity of larger drainage structures.
Note that the goal of our experiments is not to provide a detailed analysis of the precise dependency of the complexity of flow structures on n, or to give an extensive evaluation of the complexity for landscapes of all kinds of different morphologies. The main goal is to get some insight into the (in)feasibility of computing these structures exactly. Therefore we have chosen a data set which illustrates the potential blow-up well. We have repeated the same experiments for other data sets that follow the standards mentioned in Section 3.2.2. For the majority of these data sets, the numbers observed in the measured complexities differ only up to a small constant factor from the values for the `nazareth` data set (with the `nazareth` data set giving slightly higher complexities). Thus, even though for other data sets the blow-up is smaller, it will still be too costly to use exact computations when the data sets become large. For completeness we give the results for the average bit-size for one more data set, namely the `parnassus` data set—see Figure 3.4.

3.3.2 The Complexity of Watersheds, Strip Maps and River Networks

Our next step in the experimental evaluation of the exact DSD flow model is to measure the complexity of more involved drainage structures. In the following

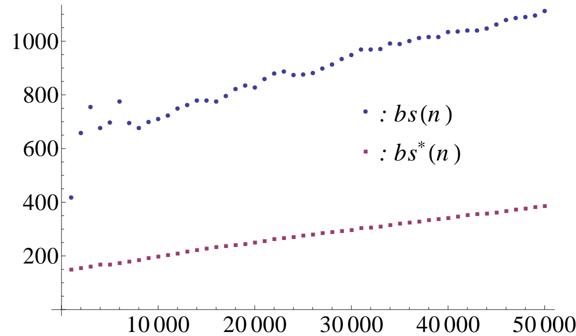


Figure 3.4 The average bit-size $bs(n)$ and average exclusive bit-size $bs^*(n)$ of the paths of steepest ascent and descent on the **parnassus** data set, as a function of the number of TIN vertices.

set of experiments we measure the combinatorial complexity and the bit-size of watershed maps and river networks on TINs. Recall that to compute the watershed map we have to construct first a more refined intermediate structure, the *strip map*. We already mentioned that the boundaries between incident watersheds in the watershed map is a subset of the paths constituting the strip map. Thus, although the strip map is not by itself a structure that is used in hydrological applications, its complexity provides further insight into the computational effort that is needed for computing exact watersheds on a TIN. For this reason, in this set of experiments we also measure the combinatorial complexity and the bit-size of the strip map. With the results of these experiments we intend to provide a clear picture of the computational demands of the studied flow model along with the restrictions on the size of input data that can be processed.

For the current set of experiments we have used the same 50 TIN instances derived from the **nazareth** data set that were used for the experiments in Section 3.3.1. Recall that these TINs consist of from 1000 up to 50,000 vertices and are constructed using a greedy method that tries to minimize the elevation difference between the TIN surface and grid points of the complete data set. For each of these TIN instances we computed the combinatorial complexity and the bit-size of the watershed map, the strip map, and the river network of the TIN. (Here we only count the number of edges, and the bit-size needed to represent their coordinates; the pointers needed for the graph structure (edges, etc) are not taken into account to avoid dependency on implementation details.) The results of these experiments are depicted in Figs. 3.5 and 3.6. We also provide the results of the same experiments applied to TINs constructed from the **parnassus** data set—see Fig. 3.7.

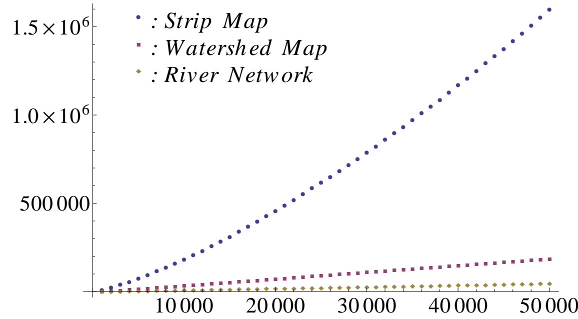


Figure 3.5 The combinatorial complexity of the strip map, watershed map, and river network for the `nazareth` data set, as a function of the number of TIN vertices.

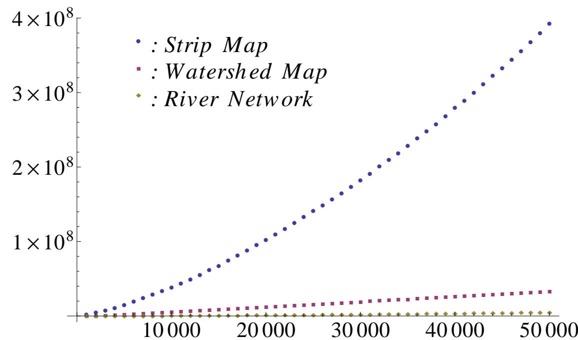


Figure 3.6 The bit-size of the strip map, watershed map, and river network for the `nazareth` data set, as a function of the number of TIN vertices.

Discussion The most striking result from the experiments is the large difference between the complexity (and the bit-size) of the strip map, which is just an intermediate structure, and the complexity (and the bit-size) of the watershed map and river network. Indeed, while the combinatorial complexity of the strip map is much higher than that of the TIN—given the results of the previous subsection, this was to be expected—the complexity of the watershed map is comparable to that of the TIN, and the complexity of the river network is even smaller. When one considers the bit-size instead of the combinatorial complexity, there is a small increase for the watershed map and river network, as compared to the TIN. Still, the bit-size for the watershed map is no more than twice the bit-size needed for the representation of the input (and the river network is even smaller). The bit-size

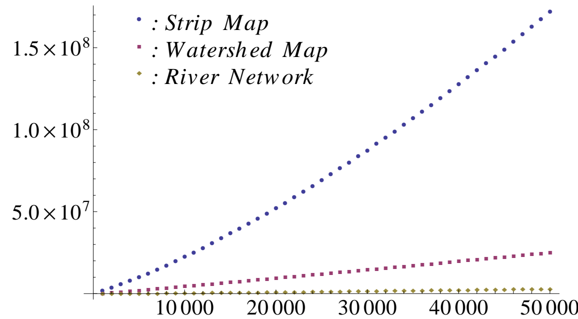


Figure 3.7 The bit-size of the strip map, watershed map, and river network for the `parnassus` data set, as a function of the number of TIN vertices.

of the strip map, on the other hand, grows to approximately 400 MB for the TIN with 50,000 vertices. (Note that the size of the strip map is even higher than the total size of the exclusive paths from the vertices, because the strip map includes all *locally* steepest paths, so that more paths may emanate from each vertex.)
We conclude that the construction of the strip map is a major computational bottleneck when computing watersheds or river networks on a TIN using the DSD model. In fact, explicit computation of the strip map is prohibitive for large TINs. Indeed, one of the reasons that we chose to conduct experiments for the presented range of input sizes is because we experienced problems with the main memory usage when processing larger TINs: the 3.4 GB of RAM of our workstation proved to be barely sufficient for computing the strip map for TINs of 200,000 vertices, let alone the extended time needed for these computations. On the other hand, the watersheds and river networks themselves have reasonable complexity, even when computed exactly and the bit-size of their exact coordinates is taken into account. This raises two questions. First, given the fact that computing watersheds exactly by computing the strip map is infeasible for large data sets, what is the quality of known heuristics for computing these structures? Second, is it possible to compute watersheds exactly without explicitly computing the strip map as an intermediate structure? We study these two questions in the next sections.

3.4 Quality of Inexact Flow Models

The DSD model treats the TIN as a continuous surface, where water can flow along edges as well as across triangle interiors. We saw in the previous section that performing exact computations based on the DSD model is expensive. Discrete flow models, where water is propagated only through a fixed network (such as the

TIN edge set), avoid the combinatorial blow-up and bit-size problems and are thus potentially much more efficient. However, the output induced by such a model may contain inconsistencies with the DSD assumption. Thus we would like to assess the quality of the output of these approximate methods.
So far, the only attempts to measure the output quality of a flow model were based either on visual criteria (which is rather subjective), or on a comparison with output induced by a higher resolution data set (which probably tells us more about the quality of the low-resolution data set than about the flow model). The availability of an exact implementation of the DSD model provides us with a point of reference to evaluate the quality of the output of the inexact but more efficient discrete flow models: for small to medium size TINs, we can compare the output of the exact DSD model with the output of the discrete flow models and see then which discrete model gives the best approximation to the DSD model.
We note here that using the exact DSD model as a point of reference does not imply that the DSD model always produces results that represent the flow of water on the real terrain accurately. However, the discrete flow models are all based on the assumption that water follows the direction of steepest descent; they just approximate the computation of flow for efficiency reasons. Hence, it makes sense to compare their output to the exact DSD model.
We have selected three popular methods for computing watersheds on TINs using a discrete flow model. The first method is similar to the algorithm proposed by Mangan and Whitaker [60], and involves expanding approximate paths of steepest descent from TIN vertices following only the edges of the TIN. We refer to this method as the *steepest-neighbour method*. The second method considers a space-sweep mechanism for the computation of watersheds [92]. We call this method the *simulated immersion* method, following the original term from the literature. The third method considers computing the boundaries of watersheds by expanding descending and ascending paths of TIN edges from the saddle vertices of the terrain [87]. We refer to this method as the *boundary-expansion* method. These three methods are very efficient in terms of memory usage and computational time. For all these methods, watersheds consist of only full triangles, unlike watersheds according to the exact DSD model where watershed boundaries could extend also through triangle interiors. Hence, the combinatorial complexity of the watershed maps computed by these methods is the same as the size of the input. Also, the bit-size of the computed watershed maps is basically the same as the bit-size of the input TIN; this is because we only need a few extra bits for each vertex, edge and triangle of the TIN to indicate the watershed that it belongs to. Next we provide a more detailed description for each of these heuristics.

- *Steepest-Neighbour Method:* This method first assigns each TIN vertex to a local minimum by expanding an approximate path of steepest descent from this vertex, as follows. Consider a vertex v of the input TIN $\mathcal{T}$ and let $E(v)$ be the set of edges in $\mathcal{T}$ incident to v. The assumption is that water from a vertex v flows through the edge $e \in E(v)$ with the steepest descending slope.

Hence, starting from v, we expand a path by picking recursively the steepest descending edge incident to the current path vertex until we encounter a local minimum v_m. Then we assign v, and all the other vertices on this path, to the watershed of v_m. After assigning each vertex of $\mathcal{T}$ to a local minimum according to this procedure, we then determine which triangles are included in the watershed of each minimum. A triangle t is included in the watershed of local minimum v_m if at least two of its incident vertices drain to v_m. If all vertices of t drain to different minima then we assign t to the local minimum that receives water from the vertex of t with the smallest elevation.

- *Simulated Immersion:* This method assigns the TIN vertices to the watersheds of the terrain's local minima using a space-sweep technique. Intuitively, we start from the lowest vertex of the TIN and as we move upwards we construct the terrain watersheds by labeling the vertices that appear on the contours of the TIN. More formally, we first sort the TIN vertices in order of increasing elevation. We then scan the ordered sequence of vertices starting from the lowest vertex. Each time we encounter a local minimum v_m, we assign v_m to its own watershed. Each time we encounter a vertex v that is not a local minimum we look at the neighbouring vertices of v with smaller elevation than v. If all neighbours are assigned to the watershed of a local minimum v_m then we assign also v to v_m. If not all neighbours are assigned to the same minimum, in our implementation of the method, we search for the local minimum to which the largest number of neighbours of v are assigned and then we assign v to the watershed of this minimum.

- *Boundary Expansion:* In this method, the watershed boundaries are outlined by expanding ascending and descending paths from the saddle vertices of the TIN. These paths are sequences of TIN edges that connect saddle vertices with local minima and local maxima on the terrain. Thus, the method is somewhat similar to the exact method of Liu and Snoeyink [57], except that paths are only expanded from saddles and that paths are restricted to TIN edges. For more details on how these paths are expanded in the vicinity of each saddle vertex, the reader may refer to the work of Edelsbrunner *et al.* [32]. The TIN is subdivided by the computed paths into regions where, hopefully, each region contains only one local minimum. The triangles that are contained in such a region then form the watershed of this minimum. However, it is not always guaranteed that each of the delineated regions contains exactly one local minimum. In our implementation, in the case that more than one minimum appear within the same region R, we assign a triangle t to the watershed of a minimum $v_m \in R$ if (most of) the vertices of t are closer to v_m than to any other minimum in R. In the case that no minimum exists within R, the triangles within R are not assigned to any watershed.

To compare the quality of these three heuristics, we compute the watershed map

of a given TIN four times: once using each heuristic, and once using our exact implementation of the DSD model. Then, for each heuristic, we compute the percentage of the TIN area that is assigned by this heuristic to the same watersheds as in the DSD model. We conducted this experiment using three different TIN data sets, `nazareth`, `duchesne` and `parnassus`, each consisting of 50,000 vertices. These TINs represent different types of landscapes, so that we can get a first impression of whether the performance of the heuristics is affected by the terrain morphology. (To draw firm conclusions about this, a more extensive investigation would be needed.)

Most available digital terrain data sets contain many spurious minima. These minima induce very small watersheds in the watershed map, which do not substantially affect the flow of water on the TIN surface. For this reason, it is common practice in hydrological applications to merge these small watersheds into larger ones. This process is called *hydrological conditioning* [28] and there exist many different approaches to do this. In most cases, after computing the watersheds of all minima on the initial terrain model, watersheds are classified as either significant or insignificant, depending on some geometric characteristic like the *topological persistence* of the watershed [71] or the watershed *area measure*, and then insignificant watersheds are removed by merging.

We have measured the performance of the flow heuristics both before and after the conditioning. Although the watershed subdivision after the removal of the spurious watersheds is what is sought in practice, we also measured the performance of the examined heuristics considering all the minima. This is because the performance of a method that computes watersheds at this stage influences the decisions taken during the conditioning: if some method assigns large regions of the terrain to watersheds of the wrong minima then a wrong set of minima will be considered as insignificant and thus will be removed during the unification process. The criterion that we used for deciding which minima are spurious is the topological persistence of each watershed. The topological persistence of a watershed is the elevation difference of its minimum and its lowest saddle point, where water would spill over into a neighbouring watershed. Watersheds with the smallest topological persistence got merged with the neighbouring watershed into which water would leak from the lowest saddle point, until 30 watersheds were left.

For the heuristics, we consider two variants of the conditioning algorithm based on topological persistence, which differ only in the way in which the lowest saddle vertex is determined for each watershed:

- In the *standard method* we simply use for each watershed the lowest saddle point on its boundary.
- In the *hybrid method* we proceed as follows. The watersheds which are computed by the heuristics (usually) have different boundaries than the watersheds computed with the exact method and therefore they are not adjacent to the same saddle vertices. Thus, the persistence value of a watershed computed by the heuristic can be different from the persistence value of the

corresponding watershed in the exact method. The idea is now to use the persistence value of the exact method when we do the conditioning on the heuristically computed watersheds. We do not want to compute the exact watersheds to determine these values, but, fortunately, this is not necessary: we can just expand paths of locally steepest descent from each saddle, using the exact flow model, determine which local minimum is reached from the saddle, and then associate the saddle to the watershed of that local minimum. Thus, a saddle may be associated to a watershed even if it does not lie on its boundary, as computed by the heuristic.

The results of the experiments are shown in Table 3.2. A toy example of the output of the exact method and the three heuristics is illustrated in Fig. 3.8.

Table 3.2 Performance of the heuristics (SN = Steepest neighbour, SI = Simulated Immersion, BE = Boundary Expansion), measured as the percentage of TIN surface area that is assigned to the same minima as in the DSD model, before and after conditioning.

	SN	SI	BE
before conditioning			
`nazareth`	74.6	57.1	78.1
`duchesne`	77.9	52.6	73.0
`parnassus`	83.6	61.7	82.3
after conditioning: standard / hybrid			
`nazareth`	54.6 / 95.1	48.4 / 90.5	44.0 / 92.5
`duchesne`	93.7 / 95.9	84.9 / 94.5	86.0 / 90.8
`parnassus`	93.3 / 96.8	78.9 / 93.8	80.1 / 91.4

Discussion Before conditioning, the immersion method gives the worst results. The other two methods are comparable, both having overlaps with the exact method which are roughly 70% – 80%. After conditioning using the standard method, the steepest-neighbour heuristic gives consistently the best results. For the `duchesne` and `parnassus` data sets the performance of this method is around 93% which is a good approximation of the exact result. However, all heuristics present a very poor performance for the `nazareth` data set; the performance is close to 50% for each heuristic. This is even worse than the performance of the heuristics before conditioning. This is possibly due to the fact that the `nazareth` data set represents a terrain with a small variation in the elevations of the vertices and many spurious watersheds. Small changes in the boundaries of the outlined watersheds may lead to the computation of different persistence values for each minimum, which in turns results in a different sequence of watershed unification operations.

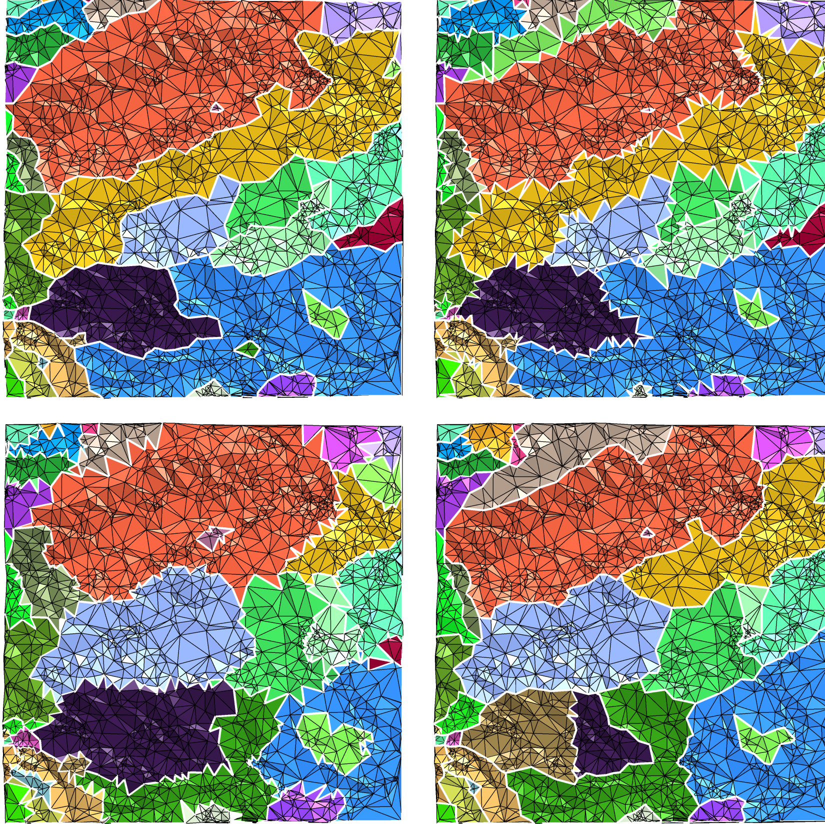


Figure 3.8 Watershed for the `parnassus` data set as computed by the various algorithms (standard method, before conditioning): exact (top left), steepest-neighbour (top-right), immersion (bottom-left), boundary expansion (bottom-right). Watershed boundaries are indicated in white. For the heuristics sometimes a watershed of a local minimum is reduced to single vertex or edge and thus it does not appear to occupy any area.

Conditioning according to the hybrid method leads to a very good performance for all examined heuristics; all heuristics lead to watershed maps that have roughly an $90\% - 96\%$ overlap with the one computed by the exact method. Still, the steepest-neighbour method yields slightly better results than the other two heuristics.
The question is, of course, which price do we pay for using exact computations to assign saddles to watersheds in terms of computation times. The next table

shows that the price is small: the overall computation times (computing initial watersheds + conditioning) for the hybrid methods is only about 2% – 11% more than the computation times when we use the standard method for conditioning.

Table 3.3 Computation times in seconds of the heuristics for computing watersheds, and of the conditioning algorithms.

	SN	SI	BE
computing initial watersheds			
`nazareth`	15	15	46
`duchesne`	14	14	45
`parnassus`	14	14	45
conditioning: standard / hybrid			
`nazareth`	112 / 117	113 / 116	115 / 117
`duchesne`	117 / 134	121 / 135	128 / 138
`parnassus`	64 / 72	67 / 74	67 / 76

3.5 An Output-Sensitive Algorithm for Computing Watershed Maps

In the previous section we studied various heuristics for computing watersheds. The resulting watersheds were clearly different from those computed by the exact algorithm, so the question arises whether one can compute the exact watersheds in a more efficient manner than by first computing the strip map. Recall that the strip map is just an intermediate structure and it has much higher complexity than the final watershed map. In this section we therefore design an *output-sensitive* algorithm: an algorithm whose running time depends on the size of the watershed map itself and not on the size of the strip map.

In Chapter 2 we described an efficient mechanism that extracts important information related to drainage structures on TINs without computing these structures explicitly. More specifically, we described a technique that can expand $\Theta(n)$ flow paths on a TIN, without computing all the intersection points of the paths with the TIN edges. Using suitable data structures, the algorithm can be made to run in $O(n \log n)$ time. This is possible since we treat paths as piecewise linear curves on the surface of the terrain; these curves are evaluated only at selected points using linear functions defined by the triangles of $\mathcal{T}$. We showed that this mechanism can be used to compute, in $O(n \log n)$ time, for each local minimum the set of triangles that are *fully* included in its watershed, or the surface network (see Section 1.1 for a definition) of the terrain. It should be emphasized here that this running-time analysis is done in the standard way, that is, it considers numerical operations as taking unit time—it does not consider the bit-sizes needed to do the computations exactly. We will come back to this issue in more detail later.

We will show that the same basic mechanism can be used to obtain an output-sensitive algorithm for computing the watershed map on a TIN. The key idea is to examine adjacent strips before computing their boundaries explicitly. We call two strips in the strip map *adjacent* if they share a common boundary. If two adjacent strips belong to different watersheds then their common boundary is a watershed boundary and therefore it also appears in the watershed map. Thus, we will use the implicit mechanism to find out first which strips boundaries/flow paths appear as watershed boundaries in the watershed map and then expand only those paths explicitly. We next provide a more detailed description of the algorithm.

The boundary of a strip consists of a segment of a valley edge, a segment of a ridge edge and paths of locally steepest ascent/descent that emanate from TIN vertices on the sides of the strip; the endpoints of the valley edge segment and of the ridge edge segment are the points where the side paths hit these edges for the first time. (Recall that a valley edge is an edge such that the DSD in the interior of both of its incident triangles points towards this edge. Similarly, a ridge edge is an edge such that the DSD in the interior of both of its incident triangles points away from the edge.) We call the endpoints of the valley edge segment of a strip the *foot points* of the strip and we call the endpoints of the ridge edge segment of the strip the

head points of the strip.
Two strips are adjacent if and only if they share a common side path, or (part of) the same valley edge segment, or (part of) the same ridge edge segment. Two strips share a common side path only if they share a common terrain vertex. Thus, for a strip s we can find which other strips are adjacent to s if we examine the boundaries only around the TIN vertices incident to s and the foot points and head points of s.
To find out which strips constitute the strip map, we cannot afford to compute the strips explicitly. Instead, we seek the TIN vertices incident to each strip and compute the head points and foot points of the strip. Therefore, for each vertex $v \in \mathcal{T}$ consider the set of triangles in $\mathcal{T}$ incident to v. We call these triangles the *star* of v. For all the paths of locally steepest descent and all the paths of locally steepest ascent that emanate from v, we compute only the part of these paths that extends through the star of v. Computing these path segments for each vertex on $\mathcal{T}$ takes $O(n)$ operations in total. The path segments that we constructed around v subdivide the star of v into regions, where each region belongs to a different strip. To compute the foot points and the head points of each strip we expand the rest of the paths that we initiated around each vertex, but this time we do this implicitly, using the mechanism described in Chapter 2. This takes $O(n \log n)$ numerical operations in total. We label each expanded path with the two strips that share this path on their boundary. This way, after computing the foot points and head points, we can find out exactly which paths represent the boundary of the same strip by just traversing the foot and head segments. This takes $\Theta(n)$ operations for all strips in total. By traversing these segments and given the strip labels of each path, we have also computed for each strip all the strips that are adjacent to it.
Next we compute for each strip the local minimum whose watershed contains this strip. To do this we pick an arbitrary point in the interior of each strip; for instance we can pick a point in the interior of each strip's foot segment. We then expand the trickle path from each of the selected points simultaneously, using the implicit expansion mechanism. This takes $O(n \log n)$ operations in total. Thus we compute for each of these paths the local minimum where the path ends. We then label the strip from which the path came with the local minimum where the path ends. Then, having labelled all the strips, we look at their common boundaries. If two adjacent strips share a common path boundary and are labelled with different minima, we expand this path explicitly. If two adjacent strips labelled with a different minimum share a common head segment we just mark this segment as part of the boundary of their watersheds [2]. This process takes $\Theta(k)$ numeric operations in total, where k is the total combinatorial complexity of all the explicitly expanded paths. From the above it follows that $O(n \log n + k)$ numerical operations are sufficient for the execution of the entire algorithm. From the anal-

[2] Two adjacent strips that share a common foot segment are always part of the same watershed. Yet, we compute the two foot points of each strip anyway to keep track of the topology of the strip.

ysis also of the implicit expansion mechanism (see Section 2.3), we conclude that during this process we have to compute at most $O(n+k)$ path points.
As we mentioned already, we measure the computational complexity of the algorithm as the total number of numerical operations that are carried out by the algorithm. This does not include the bit-sizes of the numbers that are handled in these operations. The presented mechanism may not avoid to compute points of large bit-size. However, the main goal of this mechanism is to reduce the combinatorial size of the computed data; as we expand a set of flow paths, the goal is to compute fewer path points than if we naively constructed the complete representation of these paths. As we observed from the experiments in Section 3.3.2, the combinatorial complexity of the strip map of a TIN seems to grow like a superlinear function of the input size on adversarial data sets. Thus, the average number of paths that intersect a terrain edge is not a constant but grows as the input size increases. We expect that, in practice, the presented algorithm will compute, on average, only a constant number of intersection points per edge. This can be only verified by implementing this mechanism and conducting experiments similar to the ones of Section 3.3.2.

3.6 Concluding Remarks

We presented the first implementation of an algorithm that computes watersheds on a TIN following the exact DSD model, that is, where water always follows the direction of steepest descent on the TIN surface. Since the algorithm needs exact arithmetic, it is a rather costly process because it first computes the strip map and because the exact computations need very large bit-sizes for the coordinates. Hence, the algorithm cannot be used on large data sets. However, the implementation allowed us to investigate to what extent the output of existing heuristics is consistent with the exact flow model. Of the heuristics we investigated, the one proposed by Mangan and Whitaker [60] performs best. In practice, one often applies conditioning to get rid of small watersheds. We showed that doing this using a hybrid method, which assigns saddles to watersheds using exact computations, produces very good results while being almost as fast as the standard method. Hence, we feel this is a good approach to use in practice. Finally, we presented an exact algorithm for computing watersheds that avoids computing the strip map as an intermediate structure. We leave the implementation of this new algorithm, and the investigation of its running time and memory usage in practice, for future research.

Chapter 4

Identifying Watersheds on Noisy Terrains

4.1 Introduction

In the previous chapters of this thesis we considered several problems that arise when computing drainage structures on triangulated terrain models, problems that arise in theory but also in practice. So far we examined the high complexity of drainage structures themselves, whether this is the worst case combinatorial complexity of these structures in theory or the infeasibly large bit-sizes of their explicit representations in practice. In the present chapter though we will investigate algorithmic problems that appear in the computation of flow structures when noise exists in the input data. We continue with a few definitions, already provided in other chapters of this work, which are important for the problems described in this chapter also.

As already presented in this thesis, the most natural model for water flow on TINs is that water follows the direction of steepest descent (DSD) on a surface. As water flows across the surface of a terrain $\mathcal{T}$, following the DSD, it accumulates in the local minima of $\mathcal{T}$. For a local minimum p on $\mathcal{T}$, the *watershed* of p is the set of all points on $\mathcal{T}$ from which water flows down to p as it follows the DSD. If we delineate the boundaries of the watersheds for all local minima of $\mathcal{T}$ then the induced subdivision is the *watershed map* of $\mathcal{T}$. The works of de Berg *et al.* [6], Yu *et al.* [96] and McAllister [2] provide an analysis of the worst-case complexity of the drainage structures that may appear on TINs according to the DSD model. They show that, due to the assumption that water follows the direction of steepest descent, flow paths and watershed boundaries on the surface of the TIN can run across the interiors of triangles. In Chapter 3 we described that in practice this may cause robustness problems; Liu and Snoeyink [57] show that as a flow path follows the DSD on the surface of a TIN, fixed bit-size numeric values are not

adequate to represent the coordinates of an intersection point of the path with a terrain edge. Indeed, most existing software packages for flow computation on TINs do not follow the exact DSD model, but discretize the flow, for example by only allowing water to move between a fixed set of points like the vertices of the TIN or the barycenters of the triangles [28].

However, other than the use of appropriate arithmetic, there are other factors that affect the output of a flow model. One such factor is the existence of noise in the input data. To explain better the concept of noise for TINs, consider the vertices of such a terrain. These vertices are supposed to represent points on the actual surface which is approximated by the TIN. However, this is rarely the case; the coordinates of the TIN vertices do not match with points on the actual surface. This noise, that is this inaccuracy in the coordinates of the vertex set, is a result of several processing stages. These may include inaccurate field measurements, the possible use of an interpolation technique to construct extra points, conversions between different terrain models and others. To model noise, we consider that the TIN is *imprecise*, that is the coordinates of its vertices do not have fixed values. Yet, for simplicity reasons, imprecision is usually considered only for the elevation of those vertices; the xy coordinates of each vertex are fixed but the elevation of the vertex is represented by an interval of possible values. If for a TIN $\mathcal{T}$ we pick for each vertex a specific value within its elevation interval then we get a *realisation* or a *perturbation instance* of $\mathcal{T}$.

Distinct realisations of an imprecise terrain may differ considerably in their drainage properties. For instance, consider the structure of the watershed map of an imprecise TIN. Looking at different realisations of the TIN, the boundaries between watersheds may change substantially, and watersheds may even disappear or appear as the set of local minima may not be the same among the distinct realisations—see Fig. 4.1 for an example. Yet, the drainage characteristics on some regions of the watershed map may remain more or less the same among the different perturbation instances. Hence, it is very interesting to examine which regions of the watershed map are more "stable", that is which regions maintain their drainage chracteristics without being affected by noise. Given a discrete set of distinct realisations of an imprecise TIN, can we define a method for identifying which watersheds correspond to each other among the different realisations? An answer to this question may also be useful for applications of *data conflation*, that is, for identifying the same watershed(s) between TINs that represent the same geographical area but were generated from different data sources.

Another problem which is inherent in delineating watersheds on terrains is the existence of spurious minima. In most of the available digital terrain data sets there exists a large number of local minima that correspond to watersheds of very small area [46]. These minima are either artefacts induced by the noise in the data, or they represent real-world entities that do not significantly affect the drainage attributes of the terrain, especially when looking on a larger scale. Hence, it is important for hydrological applications to provide a mechanism that merges small watersheds into larger entities. This leads to a more realistic representation of the

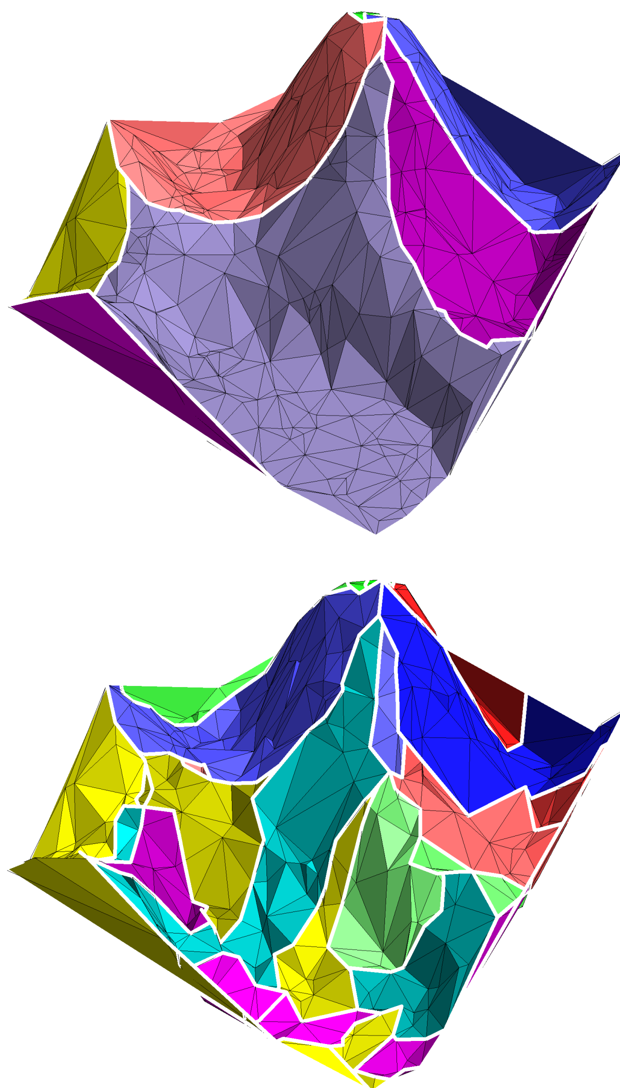


Figure 4.1 The watershed maps of two different realisations of an imprecise TIN.

drainage properties of the real-world terrain. This process is known as *hydrological conditioning* and has attracted a lot of attention within the fields of hydrology and GIS. Filtration of insignificant watersheds is also an important issue in remote sensing [67] and image analysis [55], where gray-scale images are treated as DEMs with brightness being interpreted as elevation. One popular conditioning method involves lowering the terrain elevation on ridges that separate local minima so

that drainage outlets are created between the corresponding watersheds [17]. But for most cases in practice, watersheds are merged using a flooding process, where basins are flooded symbolically until they overflow into a neighbouring basin, with which they are then merged into one entity. In fact, there exist many variants of these two watershed filtration mechanisms. Collischonn *et al.* [23] compare the performance of different filtration methods on DEMs by considering the river network on the conditioned terrain. Revsbæk [71], and Liu and Snoeyink also consider different geometric criteria to define insignificant watersheds. Considering noise in the elevation data, a question that arises is: which watershed merging variant provides the most consistent results among different realisations of the same terrain?

Our results In the current chapter we introduce and evaluate two different techniques for matching watersheds across different realisations of a triangulated terrain. We study the performance of these techniques when combined with two different strategies for merging insignificant watersheds. For the evaluation of our methods, we used a robust C++ software package that computes drainage structures on TINs; we already provided a brief description of this package in Section 3.2.1. Our implementation follows the formal flow model described by de Berg *et al.* [6] and Yu *et al.* [96], that is based on the exact DSD model. Recall that our implementation is also the first of this kind to support the use of exact arithmetic.
We have used our implementation on TINs that represent real-world geographical areas to perform experiments involving perturbations of the elevation values of their vertex sets. As a first step, for each terrain instance we test two different methods for merging spurious watersheds into larger ones. We use the output of these methods as an input to two matching algorithms that attempt to identify the same watershed entities across different perturbation instances of the same terrain.

4.2 Description of the Main Algorithms

4.2.1 Algorithms for Merging Watersheds

Before we attempt to identify watersheds among different realisations of imprecise terrains, we first have to resolve another important issue. For most of the digital terrain data sets that are available today, a considerable part of the terrain surface is covered by watersheds that have relatively small area. In practice, such watersheds are not considered to have a substantial impact on the drainage properties of the landscape; in case of heavy rainfall shallow pits will become flooded and the water that they accumulate will subsequently flow towards larger basins. To some extent, this interpretation is meaningful, depending also on the scale that we use to examine a given surface. Such small watersheds appear in large numbers

in most terrain data sets and thus it becomes a very difficult task to draw any conclusions on the topology of the studied terrain.

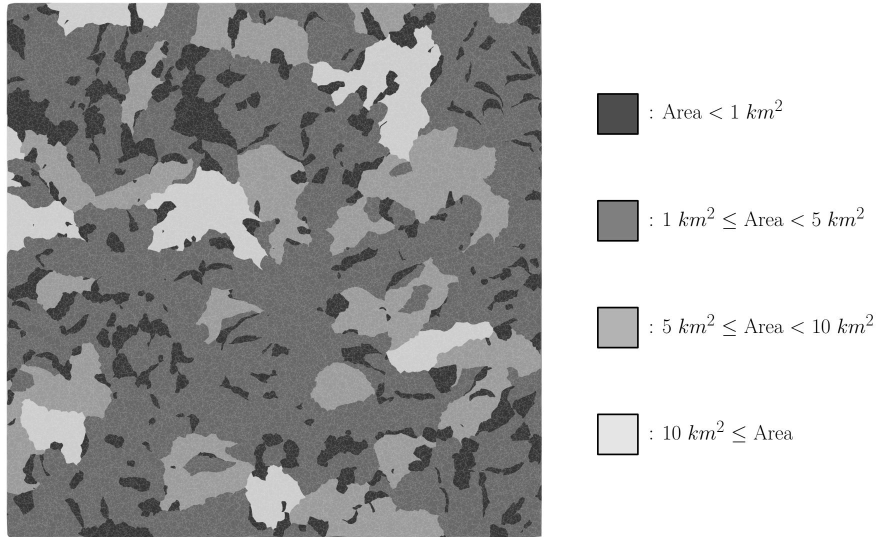


Figure 4.2 Top view of a TIN of 50,000 vertices where each watershed region is coloured according to the size of the area that it covers. Note that large connected regions of the same colour do not always constitute a unique watershed; they mostly consist of many incident watersheds that fall in the same area range.

This situation is clearly depicted in Fig.4.2. In this image, we see a top view for a TIN of roughly 100,000 triangles where each watershed region is shaded according to the area measure of this watershed. We have considered four different area-size ranges and each watershed whose area measure falls within one of these ranges is coloured with the respective shade of grey. The total number of watersheds in this terrain is 885 and approximately only 9% of the total terrain area is covered by watersheds which have a significant size.
Consider that this example is just one among different realisations of an imprecise TIN. If we pick slightly different values for the elevations of the TIN vertices, the geometry for most of the small watersheds will change substantially. Trying to find any similarities between the multitude of small watersheds among the different realisations will not produce any positive results; we would conclude that the drainage structures of the TIN differ considerably with each realisation. However, any differences in the hydrological attributes of the different realisations may appear only on a smaller scale, while on a larger scale, the flow of water is

not really affected by the displacement of a few shallow pits. Thus, it becomes clear that we first have to employ a method that merges insignificant watersheds into larger ones, so that we get a more meaningful subdivision of the terrain into drainage basins.

Next we describe a general mechanism for unifying watersheds on the surface of a terrain $\mathcal{T}$. Let $\mathcal{W}$ initially be the set of all watersheds on the surface of $\mathcal{T}$. Let m be a local minimum on the surface of $\mathcal{T}$ and let $w(m)$ be the watershed of m. Suppose that $w(m)$ is an insignificantly small watershed that cannot absorb all the water that falls on its surface. We could then assume that water accumulates in $w(m)$ until it starts leaking out of $w(m)$ from the lowest point on its boundary. Let s be this point, which is actually a *saddle* point. Consider tracing a path from s that follows the DSD on the part of $\mathcal{T}$ around s that is not included in $w(m)$. Let m' be the local minimum where this path ends and let $w(m')$ be the watershed of this minimum. We can then merge the watersheds of the minima m and m' into one watershed that we denote as $w(m', m)$; thus the watershed set of $\mathcal{T}$ becomes $\mathcal{W}' = (\mathcal{W} \backslash \{w(m), w(m')\}) \cup w(m', m)$. This process of merging insignificantly small watersheds with other watersheds is repeated until no insignificantly small watersheds remain.

We have implemented the general watershed merging algorithm by inserting the terrain watersheds in a priority queue where the head of the queue always hosts the least significant watershed of the terrain according to some significance measure. As long as the head of the queue contains an insignificant watershed, this element is extracted from the queue and merged with a neighbouring watershed w' which is selected in the manner that we described. When a watershed w is absorbed by a watershed w', we increase the significance value of the element that represents w' in the queue accordingly. Our implementation is generic as it allows the user to provide a function that measures the significance of a watershed.

For the needs of our experiments we have employed two different measures for determining the significance of a watershed:

i. *Topological Persistence:* Let m be a local minimum on the surface of $\mathcal{T}$ and let s be the saddle point that has the lowest elevation on the boundary of the watershed that corresponds to m. The topological persistence of the watershed of m is the elevation difference between the points m and s. Topological persistence is the most popular concept for measuring the significance of watersheds in GIS software [28] or of other topological notions [32].

ii. *Watershed Area:* This is simply the measure of the area of a watershed. We consider this to be an intuitive significance measure but we are not aware of any related software that makes use of this measure.

The unification process can be terminated when the least significance value of a watershed in the watershed map is larger than some predefined threshold. In our experiments we ended the unification process when the number of the watersheds was reduced to some predefined value. How to decide when to terminate this

process is an interesting question by itself and the answer may also depend on the scale on which we examine the drainage attributes of a surface. In Section 4.3 we present a simple method that can be used as a criterion for a more axiomatic selection of this threshold value.

4.2.2 Algorithms for Matching Watersheds

We now focus on two techniques for identifying watersheds across different realisations of the same terrain. More formally, consider a terrain $\mathcal{T}$ such that the elevation of every vertex of $\mathcal{T}$ is modelled as an interval of possible values. Let $\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ be a set of different realisations of $\mathcal{T}$; each realisation is created by selecting, for each vertex $v \in \mathcal{T}$, a specific elevation value from the given elevation interval of v. For each $1 \leqslant i \leqslant k$, let $\mathcal{W}_i$ be the set of watersheds that appear in the watershed map of $\mathcal{T}_i$. Our goal is to define a set of watershed entities $\mathcal{W}$ so that any element $w \in \mathcal{W}$ is represented exclusively by one element $w_i \in \mathcal{W}_i, \forall\ 1 \leqslant i \leqslant k$. In other words, if possible, we want to match each watershed of each set $\mathcal{W}_i$ with exactly one element from every other set $\mathcal{W}_j$ with $j \neq i$. We match watersheds in groups of exactly k elements, each element coming from a different realisation. We want the matched watersheds to be similar enough so as to represent the same entity, according to some notion of similarity. In the present work, we want the watersheds that are matched together to cover roughly the same xy region of the terrain; this is a strong indication that indeed, they represent the same drainage entity of the real-world terrain in the different instances of the digital model.
As we explained, there is no practical interest in attempting such a matching among terrain instances that contain a multitude of spurious watersheds. Therefore, we first apply one of the presented watershed unification techniques to the given terrain realisations, and then we use the resulting watershed maps as the input for our matching methods. Next we provide the description of each of the proposed matching methods.

Matching According to Local Minima

For this method we identify each watershed by its local minimum; watersheds that appear in distinct realisations of the same TIN are considered to represent the same entity if their local minima have exactly the same xy-coordinates.
Consider a TIN realisation $\mathcal{T}_1 \in \mathbb{T}$ and let $v_1 \in \mathcal{T}_1$ be a vertex that is a local minimum. Assume that for every $\mathcal{T}_i \in \mathbb{T}$ the vertex v_i that has the same xy-coordinates as v_1 is also a local minimum of $\mathcal{T}_i$. Let w^i_{xy} be the xy-projection of the watershed of v_i for every $1 \leqslant i \leqslant k$. We call the *core watershed*[1] of v_1 the region which is defined as $\cap_{1 \leqslant i \leqslant k} w^i_{xy}$. This region refers to the part of the terrain that drains to this specific minimum in all the given realisations; we consider this

[1] Our definition of core watersheds is partly inspired by, yet not identical to, the concept of the same name that is introduced by Driemel *et al.* [29].

to be a region that remains hydrologically stable, its drainage attributes are not affected by the imprecision in the input data. However, if v_i is not a local minimum in each realisation $\mathcal{T}_i \in \mathbb{T}$, then this region is empty.

Matching According to Area Overlap

Consider two watersheds that belong to different perturbation instances of the same TIN. If these two watersheds cover very different regions of the xy projection of the terrain, then this is an indication that they do not represent the same structure. On the other hand, if the xy regions of these watersheds overlap to a large extent, then they probably correspond to the same entity. This is the key idea on which the following method is based.

Let w_i be a watershed in $\mathcal{W}_i$, the set of watersheds in the watershed map of the realisation $\mathcal{T}_i$ of $\mathcal{T}$. Our intention is to match w with exactly one watershed from *each* other instance $\mathcal{W}_j, j \neq i$ such that the matched watersheds have the largest possible intersection on the xy-plane.

Hence, we consider the following hypergraph representation of the watersheds in the different realisations. A hypergraph consists of a set of nodes N and a set of hyperedges H, where each hyperedge h corresponds to a set of nodes $N(h)$, and a hyperedge h is said to be *incident* on a node n if $n \in N(h)$. Each node in our hypergraph corresponds to a watershed in $\cup_{1 \leqslant i \leqslant k} \mathcal{W}_i$. Each hyperedge is incident to $|\mathbb{T}| = k$ nodes, each of which corresponds to a watershed from a different realisation of the TIN. Every hyperedge is assigned a positive weight, equal to the area of the intersection of (the xy-projections of) the watersheds corresponding to the incident nodes. Our goal is to compute a maximum-weight subset $M \subseteq H$ of the hyperedges such that each node has at most one incident hyperedge in M. This problem is an instance of the problem of computing a maximum-weight k-dimensional matching in the described hypergraph [69]. Unfortunately, this hypergraph problem is known to be NP-hard and thus it is not always possible to compute an optimal solution for a large number of watersheds within reasonable time [4].

Fortunately there exists a $k + 1$-approximation algorithm based on local search, introduced by Arkin and Hassin [4]. As it is the case for local search methods, the algorithm first computes an arbitrary valid solution. A valid solution in our setting is a matching of watersheds, a set of hyperedges such that no graph node (i.e. watershed) is incident on more than one hyperedge in this set. Then, this solution is improved by repeatedly changing at most a constant number of hyperedges, until no such change can improve the total weight of the current solution further. In our implementation, we create the initial solution with the following greedy algorithm. We first sort the hyperedges by decreasing weight. Then, starting from the hyperedge of maximum weight, we scan the sorted list of hyperedges to select hyperedges for the initial solution; a hyperedge is included in the initial solution if and only if it does not share a graph node with one of the hyperedges that have already been included in the initial solution. We found that in practice, over many

executions of the approximation algorithm in our setting, the output of the local search method always turned out to be identical to the initial solution from our greedy algorithm. Therefore, in the experiments described in the next section, we only used the greedy algorithm rather than the complete local search method, which is quite slow.

4.3 Experimental Evaluation

4.3.1 A Robust Software Package for Computing Watersheds

To conduct our experiments we used a software package that computes drainage structures on triangulated terrains. In particular, we have implemented algorithms for computing paths of steepest descent/ascent, river networks (see Section 1.1.3 for a definition of this concept), watershed maps and surface networks [9, 24] on TINs, following the flow model of de Berg *et al.*. Our software was implemented in C++ using CGAL [21], which provided basic geometric objects, predicates and number types as building blocks for our needs.
As described also in Chapter 3, the implemented algorithm for computing the watershed map of a TIN $\mathcal{T}$ is based on the work of Yu *et al.*; for each vertex v of $\mathcal{T}$ we look at the slope function of $\mathcal{T}$ around v and we compute the directions of local extrema of this function. Then we expand a path of steepest ascent for each local slope maximum around v and a path of steepest descent for each local slope minimum around v. After expanding these paths for every vertex $v \in \mathcal{T}$, the terrain triangles are subdivided into facets where all the points in the interior of each facet drain to the same local minimum. Starting from the local minima of the terrain, we then tag each facet with the local minimum to which it drains. Facets that bear the same local minimum tag are part of the same watershed. After this process we delete from the skeleton of the induced subdivision all path segments that are incident to facets that both belong to the same watershed.
The user can provide his own number type implementation that will be used for the numeric computations, but we found that in practice, the algorithms of this package will not execute properly unless an arbitrary-precision number type such as `gmpq` [38, 45] is used. As explained, this is due to the fact that fixed-precision arithmetic is not enough to represent the coordinates of the intersection points between a path of steepest descent/ascent and terrain edge interiors [57]. The large bit-size of the path points that are computed during the execution of the algorithm poses a considerable restriction on the size of the TINs that can be processed by this algorithm. For more details on the high bit-size complexity of the computed structures see Chapter 3.
We ran our implementation on a Linux Ubuntu operating system version 10.10 using the GNU g++ version compiler 4.4.4 and CGAL version 3.8.

4.3.2 Experimental Setup

The TINs that we have used for our experiments are constructed from DEM data sets that are publicly available through the online U.S. Geological Survey (USGS) server (National Elevation Dataset) [90]. The data files hosted in this server appear in the `ADF` digital format, the Arc/Info binary grid format. Each of the `ADF` files that we acquired stores a grid terrain of 3,612 $\times$ 3,612 cells where the width of each cell is 30 meters and the elevation values of the cells are 4 byte floating point numbers. The maximum absolute error in the measurement of the elevation value corresponding to the centre of each grid cell is estimated at 2.5 meters for these data sets.
We have created three TINs from `ADF` files, each derived from a different DEM data set. Each of these TINs consists of 50,000 vertices and roughly 100,000 triangles. To construct each TIN, we first selected a square region of 1,201 $\times$ 1,201 cells situated in the center of the DEM. The vertex set of each TIN was created by sampling a set of points on the surface of the selected region uniformly at random. A Delaunay triangulation was then constructed on the xy-projection of the extracted point set. The final TIN can be seen as the result of lifting the vertices of this triangulation up to their original elevation values.
For consistency reasons, we used exactly the same data sets as in Chapter 3. We provide again the names and other characteristics of these data sets in Table 4.1.

Table 4.1 The data sets used in the experiments.

name	modelled region	elevation range
`duchesne`	Duchesne (UT)	[1412 m, 3737 m]
`nazareth`	Nazareth (TX)	[1051 m, 1349 m]
`parnassus`	Mount Parnassus (CO)	[1551 m, 4351 m]

The TINs that we constructed are characterized by different landscape morphologies. The `parnassus` data set represents a rough terrain surface with many peaks and ridges, while the `nazareth` data set is characterized by plateaus and relatively flat regions. The `duchesne` data set is characterized by smooth ridges.
For each of the TINs, we generated four different realisations with the vertex elevation values selected in the following manner. To the elevation of each TIN vertex, we added a value that was picked uniformly at random from the interval $[-\eta, \eta]$, where $\eta = 2.5\,\text{m}$, the maximum absolute elevation error of the DEMs from which the TINs were constructed.

4.3.3 Evaluation of the Algorithms

On top of our software that computes watersheds, we have implemented the watershed unification methods and watershed matching methods that we described in section 4.2. For our experiments we have tried all possible combinations of the

two unification methods with the two matching methods. For each of the four resulting techniques we have computed a watershed matching among the different realisations of each of the TIN data sets that we described.
In the unification stage of the experiments, we merge watersheds until there are only nine watersheds left. For both of the matching methods, we evaluate the quality of the returned matching by computing the measure of the overlap of the xy regions of the matched watersheds. We then express this quality as a percentage of the total xy area of the terrain. Thus, this percentage expresses how much of the terrain is consistently attributed to the same watershed regardless of the noise that was applied. The results of the experiments are summarized in Table 4.2.

Table 4.2 The results of the watershed matching methods when applied together with a watershed unification technique. The value that appears in each table slot is the percentage of the total xy area of the terrain that is covered by the overlap of the xy regions of the matched watersheds in the returned solution. Notation: PM:persistence-based unification with method that matches watersheds according to local minima. PO: persistence-based unification with method that matches watersheds according to their xy area overlap. AM: area-based unification with method that matches watersheds according to local minima. AO: area-based unification with method that matches watersheds according to their xy area overlap.

Name	PM	PO	AM	AO
`nazareth`	0.1%	86.0%	0.1%	47.6 %
`duchesne`	1.0%	93.4%	0.1%	53.4%
`parnassus`	29.8%	93.0%	7.3%	51.3%

Experiments With a Restricted Flow Model

We have also examined the performance of our watershed matching methods using a popular discrete flow model, that is the model where water flows only along the edges of a TIN [60]. As described also in Chapter 3, in our implementation of this model, we consider for every vertex v on the input terrain the set of edges $E(v)$ incident to v; water flows from v only along the edge $e \in E(v)$ that has the steepest descending slope among all edges incident on v. We now trace, for each vertex v, the path that water follows from v, edge by edge, until we reach a local minimum—this will be considered to be the local minimum to which v drains. Triangles are assigned to watersheds as follows: a triangle t is included in

the watershed of some local minimum p if at least two of its incident vertices drain to p. If all vertices of t drain to different minima, then we assign t to the local minimum that receives water from the vertex of t that has the smallest elevation. We have applied all four of our combined watershed matching methods using this restricted flow model on the realisations of the four TIN data sets regarded. The results of these experiments are presented in Table 4.3.

Table 4.3 The results of the watershed matching methods using the discrete flow model where flow is restricted to terrain edges. The value that appears in each table slot is the percentage of the total xy area of the terrain that is covered by the overlap of the matched watersheds in the returned solution.

Name	*PM*	*PO*	*AM*	*AO*
`nazareth`	0.1%	48.0%	53.4%	21.9 %
`duchesne`	1.0%	94.7%	14.5%	54.8%
`parnassus`	41.6%	97.2%	20.1%	69.1%

4.3.4 Discussion

From the results it becomes evident that the method that combines persistence-based unification with matching according to the overlap of watershed area has the best performance. The methods that match watersheds according to the xy coordinates of their minima consistently give bad results. This is mostly due to two reasons. First, not all vertices that constitute local minima in some realisation also appear as local minima in all other realisations. Consider a vertex v of locally minimum elevation that has a small height difference from its neighbours in the triangulation. Then even a slight perturbation among all terrain vertices may cause another nearby vertex to become a local minimum instead of v, while no large changes may be induced in the structure of the surrounding drainage area.
The second factor that impedes the performance of the minima-based method is the watershed unification process. After merging a group of watersheds into a single watershed region, there are multiple local minima within this region. In our implementation, the local minimum that represents a group of unified watersheds is always the local minimum that has the smallest elevation within the entire region of these joined watersheds. Let $\mathcal{T}_j$ be a realisation of an imprecise terrain and let v_j be a local minimum on $\mathcal{T}_j$. After applying a watershed unification process, v_j is still a local minimum of the joint watershed region in which it appears. However, even if the vertex with the same xy coordinates is also a local minimum in the other realisations of this imprecise TIN, this vertex may not always have the smallest elevation of the watersheds that it gets merged with in each other

realisation.
The matching methods that use area-based unification are outperformed by those that use persistence-based unification. In the intermediate stages of the unification process, we have observed watersheds in some of our data sets that have relatively small area (around 1% of the total terrain area) but also a very high persistence value. In fact, these watersheds appear almost unchanged across the different realisations, and when persistence-based unification is used, they appear as individual entities in the output. The area-based unification method is oblivious of such properties, and thus watersheds of this kind are absorbed during the unification process.
We see that for the best of all four methods, the discrete flow model provides a slight overestimation on the size of the stable watershed areas that are computed. The only exception appears with the `nazareth` data set, where the exact flow model yields a large percentage for this method, unlike the discrete flow model that produces poor results. Recall from Section 3.4 that `nazareth` was the only data set where the discrete flow model had a poor performance after conditioning, when computing watershed persistence values in the standard way.

4.3.5 Other Experiment Settings

In the experiments that we presented so far in this chapter, the vertices of the imprecise input TIN were picked uniformly at random from the DEM surface. The distinct realisations of each TIN were created by adding a noise value to the elevation of each TIN vertex, where the noise value was chosen uniformly at random from the interval $[-2.5m, 2.5m]$ (recall that 2.5 meters is the maximum absolute elevation error on the DEM).
To check if these settings have a considerable effect on the performance of the tested methods, we have used different settings both for building the imprecise TIN and for creating the different TIN realisations. For the resulting TIN realisations, we have conducted the same watershed matching experiments as the ones that we presented above, and we evaluated the performance of the methods under these different settings as well.
As an alternative to random sampling, we have applied a greedy method to extract the vertex set of the imprecise TIN from a DEM. The greedy method chooses the vertices to include in the TIN one by one, retriangulating the chosen vertex set after each addition, and choosing each vertex such that we minimize the maximum elevation difference between the center of any DEM cell and the corresponding point on the resulting TIN surface [65].
As an alternative to adding uniformly distributed noise to the elevations while constructing distinct realisations of a TIN, we added noise values selected according to a normal distribution with mean $\mu = 0$ and standard deviation equal to one third of the maximum absolute elevation error. Thus, there is only 0.1% probability that a vertex receives a noise value that exceeds the maximum absolute error.
The results obtained using these alternative settings are similar to those obtained

with the original settings (randomly selected vertices, uniform noise). When the imprecise TIN is built from the DEM by greedily minimizing the elevation error, rather than randomly sampling the DEM, the performance of the matching method that combines area-based unification with matching according to area overlap improves slightly. Nevertheless, also in this setting, the method that uses persistence-based unification with matching according to area overlap produces the best results.

4.3.6 Selecting a Unification Threshold

In the experiments presented in the current chapter we terminate the unification process as soon as the number of the watersheds on the terrain has been reduced to a predefined value (nine). Selecting a threshold value in this way could be considered to be a rather arbitrary decision of how many drainage basins we want to see in the terrain under study. This is similar to issues that arise in other contexts where some kind of clustering needs to be obtained: it is not always clear in advance what should be the number of clusters in the output.

To alleviate this problem, we could consider employing the matching techniques that we describe in this work to provide a criterion for selecting a threshold for the unification process. Given a set of realisations of a TIN, we can simply apply a watershed unification algorithm followed by a matching algorithm repeatedly; the first time we use a unification threshold of two watersheds, and for each subsequent iteration we increase the threshold by one. The maximum threshold that results in a matching of a given minimum quality can then be taken as the threshold value that we sought. The threshold value thus obtained may also serve as an indication of the scale on which we can examine the drainage properties of a terrain effectively using the given TIN; that is the minimum scale at which the drainage properties of the terrain are not considerably affected by noise.

4.4 Conclusions

In this chapter we evaluated the performance of different techniques for matching watersheds among distinct instances of a TIN induced by perturbations on the elevations of its vertices. We conclude that the best quality results are provided by a matching method that identifies similar watersheds based on the overlap of their xy regions. This method performs better when applied on terrains where spurious watersheds were already merged into larger watersheds according to their topological persistence values. In contrast, we showed that matching watersheds with respect to their local minima leads to bad results. These conclusions apply both for the case that water is modelled to follow strictly the direction of steepest descent on the TIN surface, and for the case that water flows only along the edges of the TIN.

It would be interesting to check the performance of the presented methods on

terrains where the uncertainty in the elevation of the vertices depends on the local landscape morphology. Another interesting problem is to develop and evaluate a method for extracting the surface network [9] from different realisations of an imprecise terrain.

Chapter 5

The Complexity of Visibility Maps on TINs Under Noise

5.1 Introduction

The applications that involve triangulated terrains, as well as other digital terrain models, are not restricted only to flow modelling. Many important problems on terrain analysis have to do with visibility. A very common problem is the following: given a point p_{view} on or above the surface of a TIN $\mathcal{T}$, we want to compute the parts of $\mathcal{T}$ that is visible from p_{view}. A point p on the surface of $\mathcal{T}$ is considered to be visible from p_{view} if the interior of the line segment defined by p and p_{view} appears strictly above $\mathcal{T}$. The set of all points on $\mathcal{T}$ that are visible from p_{view} constitute its *viewshed*. To get an adequate representation of the visible parts of the landscape, as it is seen from a viewer standing on p_{view}, we create a 2D image by projecting these parts on a plane, the *viewing plane*. The projection of the viewshed on the viewing plane is the *visibility map* of $\mathcal{T}$ induced by p_{view}.
Of course, the viewshed of a point on a TIN, and therefore the induced visibility map, may not only contain full triangles. In fact, these visibility structures can be very complex; in the worst case of a terrain of n triangles, $\Theta(n)$ thin obstacles in the foreground may appear to fragment $\Theta(n)$ long terrain edges in the background into visible and invisible pieces, resulting in a visibility map of $\Theta(n^2)$ complexity. Here, the complexity of the map is simply expressed as the number of vertices, edges and/or faces of the visible fragments of the terrain triangles as projected on the viewing plane.
The scenario that the visibility map of a TIN of n triangles has $\Theta(n^2)$ complexity is very pessimistic. The complexity of visibility maps of real-world landscapes seems to be closer to linear with respect to the size of the input. Thus it is very interesting to provide a formal argument why visibility maps of quadratic complexity do not appear in practice.

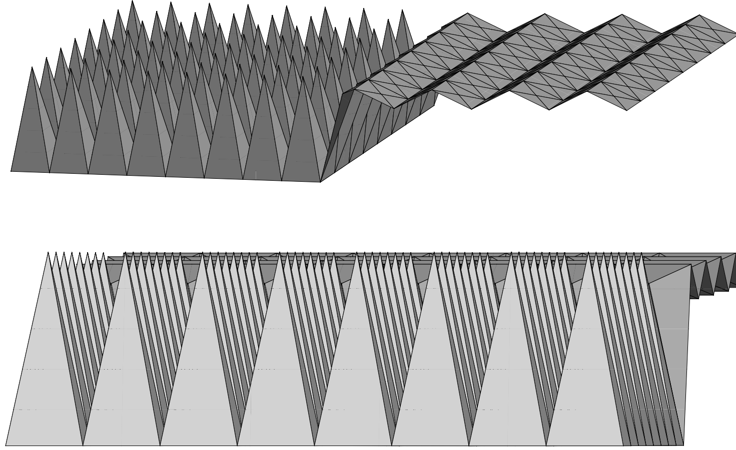


Figure 5.1 Two views of the same terrain defined by a regular grid. The second view gives a visibility map of complexity $\Theta(n\sqrt{n})$. Note that the terrain can be flattened further without changing the view combinatorially.

One possible approach to explain the low observed complexities is using a so-called *realistic input model* [11]. Here one assumes that the input has certain properties that are hopefully satisfied by inputs encountered in practice, and that rule out contrived worst-case inputs. This approach works well for many problems, and Moet *et al.* [65] have applied it to visibility maps of terrains. In particular, Moet *et al.* make the following three assumptions on the terrain: the triangles of the planar triangulation defining the terrain are fat (as defined below), the edges of these triangles differ in length by not more than a constant factor, and the domain of the triangulation is a rectangle of constant aspect ratio. Unfortunately, the assumptions do not explain why visibility maps of terrains would have near-linear complexity in practice: Moet *et al.* showed that the worst-case complexity of the visibility map of a terrain that satisfies their assumptions is $\Theta(n\sqrt{n})$. In fact, one can even assign elevations to the vertices of a triangulated grid in such a way that the triangles do not become steep while the visibility map has complexity $\Theta(n\sqrt{n})$ for certain viewing directions—see Fig. 5.1. Thus, to explain the linear behaviour, an alternative approach is needed.

Smoothed analysis. The idea behind the alternative approach is to study how sensitive worst-case inputs are to small perturbations. If a small random perturbation of the input is likely to turn any input (whether worst-case or good-case) into a good-case input, then one may argue that worst-case inputs are unlikely to be found in practice—especially if the input is subject to small measurement or

rounding errors. *Smoothed analysis* formalizes this idea.
Let $\mathcal{I}(n)$ be the set of all possible input instances (in our case: terrains) of size n. For an input $I \in \mathcal{I}(n)$, let $C(I)$ denote the quantity we want to analyse (the complexity of the visibility map). Furthermore, for any input $I \in \mathcal{I}(n)$ we define a neighbourhood $N(I) \subset \mathcal{I}(n)$ of input instances, and we define a probability distribution over $N(I)$ that indicates for every $I' \in N(I)$ the probability that perturbing the input I will result in the input I'. Now the *smoothed complexity* of an instance I is defined as

$$C_{smooth}(I) = \mathrm{E}_{I' \in N(I)}[C(I')],$$

where the expectation is according to the given probability distribution on $N(I)$. The worst-case smoothed complexity —this is what we are interested in— is then defined as

$$C_{smooth}(n) = \sup_{I \in \mathcal{I}(n)} C_{smooth}(I).$$

When $N(I)$ is defined to be the full set of possible inputs and exactly the same probability distribution is used for each I, then the above complexity measure is just the average-case complexity under the given distribution. However, it is often unclear what a reasonable probability distribution is. Moreover, average-case complexity does not indicate whether cases that are significantly worse than average may be expected to occur in practice. When $N(I)$ is narrowly defined as $\{I\}$, then the above complexity measure is just the (possibly unrealistic) worst-case complexity. By making a good choice for $N(I)$ between these extremes, one may get a more realistic estimate of the output complexity for the problem being studied.
Smoothed analysis was introduced by Spielman and Teng [82]. So far there have only been a few applications in computational geometry (see e.g. [19, 26, 27]), none of which deals with terrains.

Our results. We study the smoothed complexity of visibility maps of terrains under the following model:

- To the elevation of each vertex we add a noise value that follows a uniform distribution in an interval $[-c, c]$, where $c = c' \cdot \eta$ with η being the minimum edge length of the triangulation underlying the terrain and c' a positive parameter which we consider to be constant.

Our noise model defines for each input terrain $\mathcal{T}$ a neighbourhood $N(\mathcal{T})$ consisting of those terrain instances that can be obtained by changing the elevation of each vertex by at most c, and a probability distribution on $N(\mathcal{T})$. It is easy to see that this model alone is not sufficient to explain the linear complexity of the visibility map. Indeed, by applying a small perturbation one does not get rid of peaks that are unrealistically skinny and high, and so the smoothed visibility-map complexity of arbitrary terrains is still quadratic. Hence, we combine the power of smoothed

analysis with the ideas of realistic input models. In particular, we define the following parameters of terrains:

- *Fatness:* the smallest angle in the triangles of the underlying triangulation (or in other words, the smallest angle of any triangle's projection onto the horizontal plane);
- *Steepness:* the largest dihedral angle between any triangle and the horizontal plane;
- *Scale factor:* the length of the longest edge divided by the length of the shortest edge of the triangulation.

We assume that the fatness ϕ, steepness θ, and scale factor σ of the unperturbed terrain are constants that are independent of the number of triangles n, with $\phi > 0$, and $\theta < \pi/2$, and $\sigma \geqslant 1$. These assumptions are also used in other papers [5, 65], although the steepness assumption is not needed for the specific result on visibility maps by Moet *et al.* In her thesis, Moet [64] experimentally investigates terrain models of various mountainous regions in the US. She concludes that, at least to a large extent, they satisfy our assumptions. In itself, these assumptions do not lead to the desired result: there are terrains satisfying these assumptions with quadratic-complexity visibility maps. For example, we can take a slice of the construction of Fig. 5.1, with the aspect ratio of the domain being $\Theta(n)$. Our main result is that the smoothed complexity of any visibility map of a terrain satisfying the abovementioned assumptions is only $\Theta(n)$. This result can be generalized to certain non-uniform noise distributions. We also prove $O(nk)$ smoothed complexity for the visibility map of terrains that contain k triangles that do not fulfil our assumptions; the bound becomes $O(k^2 + n)$ when additionally the xy-domain of the terrain is a square. To avoid technical details regarding what happens if one looks at the boundary of the terrain from the side, we focus on the case of perspective views with the view point being located above the terrain.

5.2 Visibility Maps Resulting from Perspective Projection

Let $\mathcal{T}$ be a terrain with n triangles, and let E be the set of edges of $\mathcal{T}$. Let the vertices be specified by three coordinates x, y and z, where the z-axis is the vertical axis on which the elevation is specified; the x- and y-axis are orthogonal to the z-axis and to each other. Let $\overline{\mathcal{T}}$ denote the triangulation in the xy-plane defining $\mathcal{T}$. Without loss of generality we assume that the minimum edge length in $\overline{\mathcal{T}}$ is 1. Hence, the maximum edge length equals σ, the scale factor of the terrain. We assume that $\overline{\mathcal{T}}$ is a *ϕ-fat triangulation*—that is, a triangulation in which all angles are at least ϕ, for some fixed constant $\phi > 0$—and that the steepness of $\mathcal{T}$ is bounded by θ. We study the smoothed complexity of the visibility map of $\mathcal{T}$ for

perspective views, that is, the map as it appears in the projection on a viewing plane h_{view} for a given viewing point p_{view}. We assume that p_{view} is located above the terrain.

Notation, terminology and basic properties. We denote the projection of an object o onto h_{view} by $\text{pr}(o)$. For an edge $e \in E$ we use $h_{\text{align}}(e)$ to denote the plane containing e and p_{view}; thus $h_{\text{align}}(e) \cap h_{\text{view}}$ contains $\text{pr}(e)$. If e is collinear with p_{view}, there are many such planes: in that case we define $h_{\text{align}}(e)$ as the vertical plane containing p_{view} and e. The *steepness* $\theta(t)$ of a triangle t is defined as the dihedral angle of the plane containing t with the xy-plane, and the steepness $\theta(s)$ of a segment s is defined as the smallest acute angle of the line containing s with the xy-plane. Observe that the steepness of a triangle equals the maximum steepness of any segment contained in it. Recall that θ denotes the maximum steepness of any triangle in $\mathcal{T}$. The following lemma shows that the steepness of terrains that satisfy our assumptions does not change much if the vertex elevations are subject to a small perturbation.

Lemma 5.1 *Let $\mathcal{T}$ be a terrain constructed from a ϕ-fat triangulation $\overline{\mathcal{T}}$, and let θ be the maximum steepness of any triangle in $\mathcal{T}$, where $\theta < \pi/2$ is a constant. Then, after raising or lowering each vertex independently by a distance of at most c, no triangle is steeper than $\theta_{\max} = \arctan(\tan(\theta) + \frac{2c}{\sin\phi})$.*

Proof. Consider any triangle $\triangle(u, v, w)$ of $\mathcal{T}$. Then there must be a vertex of this triangle, say v, and a point p on the edge opposite to v, such that the segment pv is parallel to the direction of steepest descent on the triangle after the perturbation. Let $\overline{u}$, $\overline{v}$ and $\overline{p}$ denote the projections of u, v, and p onto the xy-plane, respectively. Since $|\overline{uv}| \geqslant 1$ and the angle at $\overline{u}$ is at least ϕ, we have $|\overline{pv}| \geqslant \sin\phi$ —see Fig. 5.2(a).

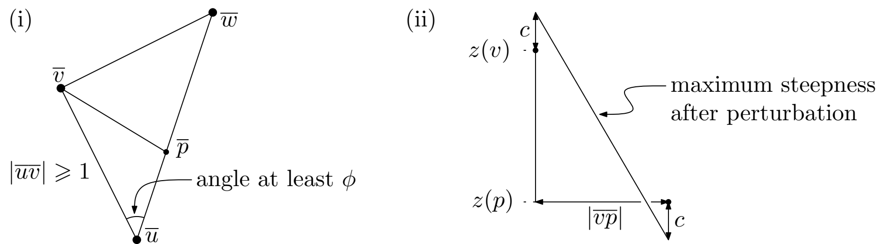


Figure 5.2 Illustrations for the proof of Lemma 5.1.

Denote the elevations of v and p before the perturbation by $z(v)$ and $z(p)$. Then,

since before the perturbation the steepness of the triangle is at most θ, we have

$$\frac{|z(v) - z(p)|}{|\overline{pv}|} \leqslant \tan\theta$$

In the worst case, the difference in elevation between v and p can increase by at most $2c$ due to the perturbation. Hence, as illustrated in Fig. 5.2(b), the steepness after the perturbation is at most

$$\arctan\left(\frac{|z(v) - z(p)| + 2c}{|\overline{pv}|}\right) \leqslant \arctan\left(\tan\theta + \frac{2c}{\sin\phi}\right).$$

□

Note that because ϕ, θ and c are constants with $\phi > 0$ and $\theta < \pi/2$, we know that $\theta_{\max}$ is a constant strictly smaller than $\pi/2$. We may assume that $\theta_{\max} \geqslant \pi/4$; otherwise we simply replace $\theta_{\max}$ by $\pi/4$ and the bounds proven in this paper will still hold.

The *perceived steepness* $\theta_{\text{view}}(e)$ of an edge e is the steepness of $\text{pr}(e)$ in the plane h_{view}. In other words, $\theta_{\text{view}}(e)$ is the smallest angle between the line containing $\text{pr}(e)$ and a horizontal line on h_{view}. Even though $\theta(e) \leqslant \theta_{\max}$ by Lemma 5.1, $\theta_{\text{view}}(e)$ may be greater than $\theta_{\max}$ and in fact be equal to $\pi/2$. Indeed, even an edge that is almost horizontal can appear vertical when projected onto h_{view}. We say that an edge e *appears steep* when $\theta_{\text{view}}(e)$ is well-defined—$\text{pr}(e)$ is not a single point—and $\theta_{\text{view}}(e) > \theta_{\max}$, otherwise e *appears flat*.

We say that an edge e lies *in front of* an edge e', if there is a ray from p_{view} that hits e before hitting e'. A *silhouette edge* is an edge e such that the two triangles of $\mathcal{T}$ that share e are on the same side of $h_{\text{align}}(e)$. Note that this is equivalent to saying that one of the incident triangles of a silhouette edge is front-facing while the other is back-facing. We say that two edges e and e' *create a visible intersection* if $\text{pr}(e) \cap \text{pr}(e')$ constitutes a vertex of the visibility map. For this to be possible, the edge hit first by a ray from p_{view}—say this edge is e—must be a silhouette edge. Otherwise it would have to be an edge on the boundary of the terrain or a non-boundary, non-silhouette edge, but both cases would lead to a contradiction: in the first case e could only be the *last* edge hit by any directed line through p_{view} because there is nothing beyond e—this is true because the domain of $\mathcal{T}$ is a triangulation and, hence, convex—while in the second case the two triangles incident to e would hide e' from view locally around the point of e' projecting onto $\text{pr}(e) \cap \text{pr}(e')$.

Above we observed that even though the edges of the terrain are not steeper than $\theta_{\max}$, they can still appear steep on h_{view}. The next lemma, which is the key to bounding the number of visible intersections, states that this cannot happen for silhouette edges.

Lemma 5.2 *Let $\mathcal{T}$ be a terrain whose triangles have steepness at most $\theta_{\max}$ and let p_{view} be a fixed viewing point. Then the perceived steepness (on any vertical viewing plane) of any silhouette edge of $\mathcal{T}$ is at most $\theta_{\max}$.*

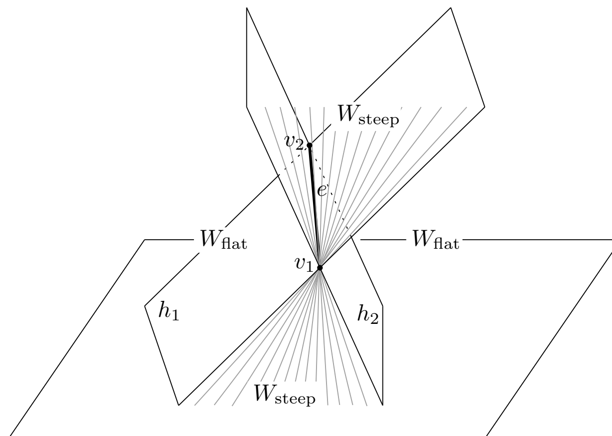


Figure 5.3 Partitioning the space around an edge in two double wedges: one containing all steep planes that contain the edge, one containing all other planes that contain the edge.

Proof. Consider an edge $e = v_1v_2$. Let h_1 and h_2 denote the two planes containing e that have steepness exactly $\theta_{\max}$. These two planes partition the space into two double wedges: $W_{\text{steep}} := (h_1^+ \cap h_2^+) \cup (h_1^- \cap h_2^-)$ and $W_{\text{flat}} := (h_1^+ \cap h_2^-) \cup (h_1^- \cap h_2^+)$, where h_i^+ and h_i^- denote the half-spaces above and below h_i, respectively. Note that W_{steep} is the union of all planes containing e that are steeper than $\theta_{\max}$, while W_{flat} is the union of all planes containing e that are less steep than $\theta_{\max}$—see Fig. 5.3.

Let $\triangle(u, v_1, v_2)$ and $\triangle(v_1, v_2, w)$ be the triangles sharing the edge e. Since no terrain triangle is steeper than $\theta_{\max}$, the vertices u and w must lie in W_{flat}. Moreover, they must lie in different parts (that is, wedges) of W_{flat}, since otherwise there would be a vertical line intersecting the interiors of both $\triangle(u, v_1, v_2)$ and $\triangle(v_1, v_2, w)$, which cannot happen because $\mathcal{T}$ is a terrain.

Now consider $h_{\text{align}}(e)$, the plane that contains e and p_{view} and whose intersection with h_{view} contains $\text{pr}(e)$. Suppose e appears steep, that is, $\theta_{\text{view}}(e) > \theta_{\max}$. Because $h_{\text{align}}(e)$ contains $\text{pr}(e)$ it is at least as steep as $\text{pr}(e)$, so we have $\theta(h_{\text{align}}(e)) > \theta_{\max}$. Since $h_{\text{align}}(e)$ contains e, this means $h_{\text{align}}(e)$ is contained in W_{steep}. Since u and w lie in different parts of W_{flat}, this implies that $\triangle(u, v_1, v_2)$ and $\triangle(v_1, v_2, w)$ lie on different sides of $h_{\text{align}}(e)$. Thus e is not a silhouette edge if it appears steep, which proves the lemma. $\square$

Counting intersections. Since the number of terrain vertices is $O(n)$, we only need to worry about bounding the number of visible intersections created by pairs of terrain edges. When two edges appear to intersect we will charge each visible intersection to the edge that is furthest from the viewer. Thus we need to count,

for each edge e in the perturbed terrain, the number of visible intersections it creates with edges in front of it. We denote this number by $K(e)$.

Consider the situation in which we have already perturbed the edges in front of e, and we wish to analyse the effect of perturbing e. Consider the triangle whose vertices are p_{view} and the endpoints of e, and define $\overline{\triangle}(p_{\text{view}}, e)$ to be the projection of this triangle onto the xy-plane. Let $E_{\text{fr}}(e)$ be the set of silhouette edges whose projection onto the xy-plane intersects $\overline{\triangle}(p_{\text{view}}, e)$, clipped to the part whose projection is contained in $\overline{\triangle}(p_{\text{view}}, e)$. We exclude the edges sharing a vertex with e; such edges cannot create a visible intersection with e. Then the visible intersections charged to e are intersections of $\text{pr}(e)$ with the upper envelope of $\{\text{pr}(s) : s \in E_{\text{fr}}(e)\}$—see Fig. 5.4(a). We denote this upper envelope by $H(e)$.

Now consider a fragment $f \in E_{\text{fr}}(e)$ that appears on this upper envelope. We wish to bound the probability that after perturbation of e, the edge e creates a visible intersection with f.

Observe that the combinatorial structure of the visibility map on a viewing plane h_{view} does not depend on the location and orientation of the viewing plane, provided h_{view} does not contain p_{view} (in which case one would not see anything). We can therefore assume without loss of generality that h_{view} is a vertical plane that contains e. For an object o, define $\text{span}_e(o)$ to be the projection of $\text{pr}(o)$ onto a horizontal line on h_{view}, and define $\text{width}_e(o)$ to be the length of $\text{span}_e(o)$. We have the following lemma.

Lemma 5.3 *Let e be an edge of $\mathcal{T}$ and let f be a fragment of a silhouette edge e' not incident to and in front of e such that $\mathrm{span}_e(f) \subset \mathrm{span}_e(e)$ and $\mathrm{width}_e(f) \leqslant \mathrm{width}_e(e)/3$. Now suppose we independently perturb the elevations of the vertices of e, where the perturbations are chosen uniformly at random from the range $[-c, c]$. Then*

$$\Pr[e \text{ creates a visible intersection with } f] \;\leqslant\; \frac{3\,\mathrm{width}_e(f)\cdot\tan\theta_{\max}}{c}.$$

Proof. Assume without loss of generality that the projection of e on the xy-plane is parallel to the x-axis. Let v_1 and v_2 be the vertices of e. Without loss of generality assume v_2 is the vertex closest to f in the projection onto the x-axis, with ties broken arbitrarily. Since $\mathrm{width}_e(f) \leqslant \mathrm{width}_e(e)/3$, the distance from v_1's projection to $\mathrm{span}_e(f)$ is at least $\mathrm{width}_e(e)/3$. We will now show that for any elevation of v_1 after the perturbation, the probability that e intersects f when v_2 is perturbed, is at most $3\,\mathrm{width}_e(f)\cdot\tan(\theta_{\max})/c$.
Let ℓ be the vertical line on h_{view} through $\mathrm{pr}(v_2)$. Assume v_1 has already been perturbed, so that v_1's position is now fixed. Consider the set of all possible positions of the projection of the perturbed vertex v_2 that induce an intersection between f and e. This set is a segment on ℓ; we denote its upper endpoint by q and its lower endpoint by r—see Fig. 5.4(b). The probability that, after perturbing v_2, the edge e creates a visible intersection with f is bounded by $|qr|/(2c)$.
We will now compute an upper bound on $|qr|$. Notice that the triangle $\triangle(v_1, q, r)$ does not necessarily contain f completely: there may be parts of f where e cannot create an intersection (for the given, fixed position of v_1), because v_2 could not be raised or lowered far enough without going beyond the given bounds on the perturbation. Let f' be the part of f inside $\triangle(v_1, q, r)$ and let v_3 and v_4 be the endpoints of f', with v_3 being the endpoint closest to ℓ. Let s be the point such that $\triangle(v_1, v_3, s)$ and $\triangle(v_1, q, r)$ are similar triangles. The line through v_1 and r has steepness at most $\theta_{\max}$ by Lemma 5.1 and the fact that h_{view} contains e; fragment f' has steepness at most $\theta_{\max}$ by Lemma 5.2. This implies that

$$|v_3 s| \leqslant 2\,\mathrm{width}_e(f')\cdot\tan\theta_{\max}.$$

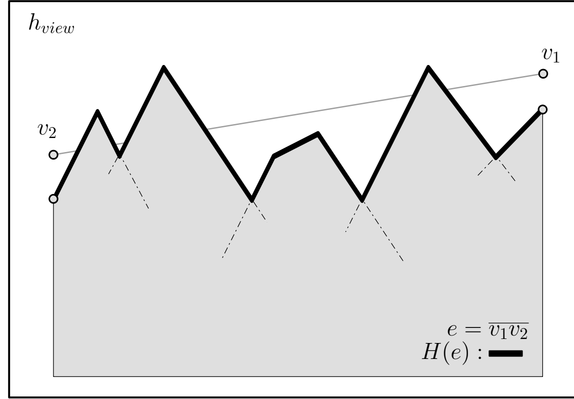
Moreover, we have $|v_1 v_3| \geqslant |v_1 q|/3$. Now

$$|qr| \;=\; |v_3 s|\cdot\frac{|v_1 q|}{|v_1 v_3|} \;\leqslant\; 6\,\mathrm{width}_e(f')\cdot\tan\theta_{\max} \;\leqslant\; 6\,\mathrm{width}_e(f)\cdot\tan\theta_{\max}.$$

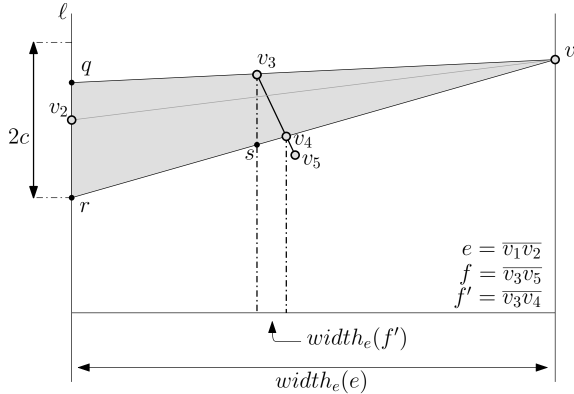
Hence, the probability of intersection between e and f for any fixed position of v_1 is at most

$$\frac{|qr|}{2c} \;\leqslant\; \frac{3\,\mathrm{width}_e(f)\cdot\tan\theta_{\max}}{c}.$$

This bound is independent of the position of v_1. Therefore the probability that, when e is perturbed, e creates a visible intersection with f is at most $3\,\mathrm{width}_e(f)\cdot\tan(\theta_{\max})/c$. □



(a)



(b)

Figure 5.4 (a) The horizon in front of e as it appears on the visibility map. (b) The shaded cone indicates all the possible positions of e that may induce a visible intersection between e and f for a fixed position of v_1. The point s is the point on the image of $v_1 r$ that has the same x-coordinate as v_3.

Lemma 5.4 *Let $K(e)$ be the number of visible intersections of an edge e of $\mathcal{T}$ with edges in front of it. Then*

$$\mathrm{E}[K(e)] \leqslant \frac{3\,\mathrm{width}_e(e) \cdot \tan\theta_{\max}}{c} + 2.$$

Proof. We distinguish two cases: either $h_{\mathrm{align}}(e)$ is vertical, or not. If $h_{\mathrm{align}}(e)$ is vertical, then $\mathrm{pr}(e)$ is vertical too. This implies $K(e) \leqslant 1$: the part of e above the intersection point would be visible and the part below it would be hidden from view.

It remains to discuss the case in which $h_{\text{align}}(e)$ is not vertical. We now need to bound the expected number of visible intersections created by e with any edge e' in front of it. Consider the situation where we already perturbed the edges in front of e, but not yet the edge e itself. Recall that e' must be a silhouette edge. Hence, we can restrict our attention to $E_{\text{fr}}(e)$, the set of segments which are parts of silhouette edges lying in front of e in the projection onto the xy-plane, excluding the edges sharing a vertex with e. We defined $H(e)$ to be the upper envelope of the projections of those segments onto h_{view} —see Fig. 5.4(a). Then we can bound $\text{E}[K(e)]$ by analysing the number of intersections of $\text{pr}(e)$ with $H(e)$.
For a fragment f on the upper envelope $H(e)$, we define an indicator random variable X_f:

$$X_f = \begin{cases} 1 & e \text{ and } f \text{ intersect} \\ 0 & \text{otherwise} \end{cases}$$

We have $K(e) = \sum_f X_f$. Observe that there can be at most two such fragments with $\text{width}_e(f) > \text{width}_e(e)/3$. Now consider the other fragments. By Lemma 5.3, we have

$$\Pr[e \text{ creates a visible intersection with } f] \leqslant \frac{3\,\text{width}_e(f)\tan\theta_{\max}}{c}.$$

Adding the at most two fragments with $\text{width}_e(f) > \text{width}_e(e)/3$, and summing over all fragments with $\text{width}_e(f) \leqslant \text{width}_e(e)/3$, we get

$$\text{E}[K(e)] \;=\; \text{E}[\sum_f X_f] \;=\; \sum_f \text{E}[X_f] \;\leqslant\; 2 + \sum_f \frac{3\,\text{width}_e(f)\tan\theta_{\max}}{c}.$$

Clearly we have $\sum_f \text{width}_e(f) \leqslant \text{width}_e(e)$, which finishes the proof. □

Using that $\text{width}_e(e) \leqslant \sigma$ and $\tan\theta_{\max} \leqslant \tan(\theta) + \frac{2c}{\sin\phi}$ (by Lemma 5.1) we obtain our final result:

Theorem 5.5 *Let $\mathcal{T}$ be a terrain of n triangles with fatness ϕ, steepness θ, and scale factor σ. Suppose we add noise to each vertex's elevation independently, where the noise value is taken uniformly at random from the interval $[-c, c]$, and c is a fixed constant fraction of the minimum edge length of the triangulation underlying the terrain model. Then a visibility map of $\mathcal{T}$ under perspective projection has smoothed complexity*

$$O\left(\left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sigma n\right).$$

Remark 5.6 The result easily generalises to other noise distributions over the interval $[-c, c]$, provided that the density function of the distribution is upper-bounded. More precisely, if the density is upper-bounded by τ, then for any

interval $[a,b] \subset [-c,c]$ the probability that the noise is in $[a,b]$ is at most $\tau \cdot |b-a|$ instead of $\frac{|b-a|}{2c}$. The bound on the probability given in Lemma 5.3 and, hence, also the final bound obtained in Theorem 5.5 then simply have to be multiplied by $2c\tau$.

Remark 5.7 One can also consider the setting where not only the elevations of the sample points are subject to noise, but also their xy-positions. In a practical setting, however, the 2D locations of the sample points determine the triangulation (because one uses the Delaunay triangulation, for instance). This means that a very different analysis will be necessary. If one ignores this aspect and simply works with the given triangulation, then our results still hold provided that (i) the noise is small enough that the fatness, steepness, and scale factor of the terrain are still bounded, and (ii) the perturbation in the xy-plane is independent of the perturbation of the elevation.

5.3 Terrains That Almost Satisfy the Assumptions

In previous works on realistic input models [11, 64] it has been observed that even in well-formed scenes, where the majority of the objects fulfills the properties of some model, there may exist a few objects that do not conform with the model. This leads us to consider the case where there exists a set of $k \leqslant n$ terrain triangles that do not follow the model assumptions. We call the triangles that have fatness ϕ, slope at most θ and edge length $\in [1,\sigma]$ *good* and the rest *bad*. Furthermore, we call a terrain edge *bad* if it is incident to a bad triangle; otherwise the edge is called *good*. We now analyse the smoothed complexity of the visibility map of a terrain with k bad triangles.

Theorem 5.8 *Let $\mathcal{T}$ be a terrain containing $n-k$ good triangles with fatness ϕ, steepness θ, and edge lengths $[1,\sigma]$ and k bad triangles. Suppose we add noise to each vertex's elevation independently, where the noise value is taken uniformly at random from the interval $[-c,c]$, and c is a constant. Then a visibility map of $\mathcal{T}$ under perspective projection has smoothed complexity*

$$O\left(nk + \left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sigma n\right).$$

Proof. For every terrain edge e we will count how many visible intersections $\mathrm{pr}(e)$ can create with edges in front of it. Recall that $\mathrm{pr}(e)$ can only create a visible intersection with the images of the segments in $s \in E_{\mathrm{fr}}(e)$ that is, the silhouette edge parts for which it holds that $\mathrm{span}_e(s) \subset \mathrm{span}_e(e)$. More precisely $\mathrm{pr}(e)$ can only create a visible intersection with $H(e)$, the upper envelope of these segments. Because $\mathcal{T}$ contains bad triangles it is possible in this setting that $H(e)$ contains steep segments, which belong to the images of bad edges.

We can partition the intersections that appear in the visibility map into two categories, which we analyze separately.

- *Category (i): at least one of the involved edges is bad.*
  Since there are at most $3k$ bad edges, and any edge can generate only $O(n)$ intersections, the number of visible intersections of the first category is $O(nk)$.
- *Category (ii): both involved edges are good.*
  In Lemma 5.4, in a setting where no bad triangles exist in $\mathcal{T}$, we proved that a good edge e can be charged with a smoothed number of at most $\frac{3\,\mathrm{width}_e(e)\cdot\tan\theta_{\max}}{c} + 2$ intersections with the segments of $H(e)$. In fact, this can be rephrased as an upper bound for the smoothed number of intersections between e and the flat segments of $H(e)$; the proof itself is independent of the existence of steep segments in $H(e)$ and thus this bound directly applies for the cases that $\mathcal{T}$ contains bad triangles. Since e is a good edge, $\mathrm{width}_e(e) \leqslant \sigma$ and e can be charged with a smoothed number of at most $\frac{3\sigma\cdot\tan\theta_{\max}}{c} + 2$ intersections with other good edges. Hence the total number of intersections between good edges is $O\left(\left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sigma n\right)$.

Summing up the intersections of categories (i) and (ii) the theorem follows. □

Fig. 5.5 shows a view of a construction where we place $\Theta(n)$ realistic triangles behind k skinny peaks in a way that the noise cannot cause their incident edges to hide behind each other. The smoothed complexity of this view is $\Omega(nk)$, thus showing that the bound from Theorem 5.8 is tight in the worst case.

A more restricted model. The result of Theorem 5.8 is a bit disappointing: even with, say, $\Theta(\log n)$ bad edges, the smoothed complexity can be superlinear. Next we show that we can get much stronger bounds if we make one more (very reasonable) assumption: we assume that the xy-domain of the terrain is a square of size $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$ Consequently, our model becomes a special case of the one presented by Moet et al. [65]. Below, we mention an interesting property that they prove for a terrain $\mathcal{T}$ that conforms with their model.

Property 5.9 *Let s be a straight line segment that intersects the projection of $\mathcal{T}$. Then s intersects $O(\sqrt{n})$ triangles of $\mathcal{T}$.*

This property is based on a packing argument that provides an upper bound for the number of good triangles that can fit in a rectangle of a certain size. The existence of bad triangles therefore does not affect this argument when applied to the good triangles. Hence, the number of good triangles intersecting any straight line segment is still $O(\sqrt{n})$.
We prove a tight upper bound for the smoothed complexity of the visibility map of such a terrain that also contains at most k bad triangles.

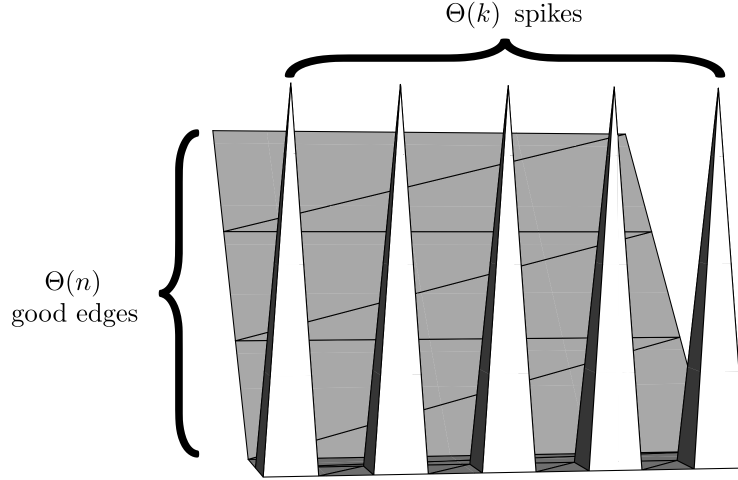


Figure 5.5 A terrain with a visibility map of $\Theta(nk)$ smoothed complexity. In this view, each of the $\Theta(k)$ thin and long spikes in the front appears to intersect with each of the $\Theta(n)$ good triangles at the back.

Theorem 5.10 *Let $\mathcal{T}$ be a terrain with an xy-domain that is a square of size $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$. Let $\mathcal{T}$ contain $n-k$ good triangles with fatness ϕ, steepness θ, edge lengths $\in [1, \sigma]$ and k bad triangles. Suppose we add noise to each vertex's elevation independently, where the noise value is taken uniformly at random from the interval $[-c, c]$, and c is a constant. Then a visibility map of $\mathcal{T}$ under perspective projection has smoothed complexity*

$$O\left(k^2 + \left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\left(\sqrt{n}k + \sigma n\right)\right).$$

Proof. As in the proof of Theorem 5.8, we distinguish several categories for the intersections of the visibility map. Next we analyse each category.

- *Category (i): intersections between a bad edge e and the steep segments of $H(e)$.*
  For an edge e, the steep segments of $H(e)$ are parts of the images of bad edges. In the worst case every bad edge may appear to intersect with $\Theta(k)$ other such edges, summing up to $O(k^2)$ intersections of this kind.

- *Category (ii): intersections between a bad edge e and the flat segments of $H(e)$.*

Let e be a bad edge. We consider first the case that $\theta(e) \leqslant \theta_{max}$. With an analysis similar to the one of ii) in the proof of Theorem 5.8, e can be charged with a smoothed number of at most $\frac{3\,\mathrm{width}_e(e)\cdot\tan\theta_{\max}}{c}$ intersections. Since $\mathrm{width}_e(e) = O(\sqrt{n})$, this number is $O\left(\frac{\sqrt{n}\tan\theta_{\max}}{c}\right)$. We examine now the case that $\theta(e) > \theta_{\max}$. There can be at most $O(k)$ steep edges that can participate in $H(e)$, that means that e can disappear behind $H(e)$ and reappear at most $O(k)$ times. Notice that e can disappear after a steep segment and then reappear after a flat segment or the other way round. Thus e can in fact participate in $O(k)$ intersections with flat segments. Hence we can have in total a smoothed number of $O\left(k^2 + \left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sqrt{n}k\right)$ such intersections.

- *Category (iii): intersections between a good edge e and the flat segments of H(e).*
  This category is identical to ii) in the proof of Theorem 5.8. Accordingly, we can have at most $O\left(\left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sigma n\right)$ intersections of this kind.

- *Category (iv): intersections between a good edge e and the steep segments of H(e).*
  We now consider the intersections that good edges induce with bad edges that appear in front of them. Let E_{bad} be the set of bad edges in $\mathcal{T}$ and V_{bad} be the set of the vertices incident to bad edges. For a good terrain edge e and a viewing plane h_{view} let $H_{bad}(e)$ be the upper envelope of the images of the bad edges that appear in front of e. We can divide $H_{bad}(e)$ into maximal polygonal chains whose endpoints are images of actual vertices in V_{bad}. Each such chain is convex and obviously $\mathrm{pr}(e)$ can create at most two visible intersections with this chain.

  For some vertex $v \in V_{bad}$, consider the ray that starts at p_{view} and passes through v. Consider on this ray the segment that starts at v and ends at the intersection of this ray with the xy-projection of the boundary of $\mathcal{T}$. We call this segment the *lifespan* of v—see Fig 5.6(b). If a good edge e intersects the lifespans of m vertices in V_{bad} then in a possible view of the terrain $\mathrm{pr}(e)$ may appear to cross all the $m-1$ convex chains between the images of those vertices plus at most two other chains. As remarked before, $\mathrm{pr}(e)$ can create at most two visible intersections with each chain. So, we have at most $2(m+1)$ visible intersections. We charge two of them to e and two to each vertex in V_{bad} whose lifespan is crossed. Hence, each vertex in $v \in V_{bad}$ can be charged with a constant number of intersections from each good edge that intersects its lifespan.

  According to Property 5.9 there can be $O(\sqrt{n})$ edges that cross the lifespan of v and thus there can be $O(\sqrt{n})$ intersections charged to v. That means that there are $O(k\sqrt{n})$ intersections in total for all the vertices in V_{bad}.

Hence, the total number of visible intersections that can appear on the visibility map is $O\left(k^2 + \left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)(\sqrt{n}k + \sigma n)\right)$.

□

5.4 Concluding Remarks

We proved that the smoothed complexity of the visibility map of not-too-steep terrains with fat triangles of similar size is $O(n)$. This is a possible explanation why in practice terrains with visibility maps of super-linear complexity are unlikely to occur. We also examined the smoothed complexity for terrains that contain a few triangles that do not satisfy the assumptions. This is the first time that realistic input models have been combined with smoothed analysis. We believe this is a promising approach, which could also shed light on the complexity of certain other structures on real-world terrains. For example, the complexity of the river network on real-world terrains seems to be linear, while the worst-case complexity of the river network on a terrain with the above-mentioned properties is still $\Theta(n^2)$ [7]. Combining these properties with a smoothed analysis may lead to better bounds.

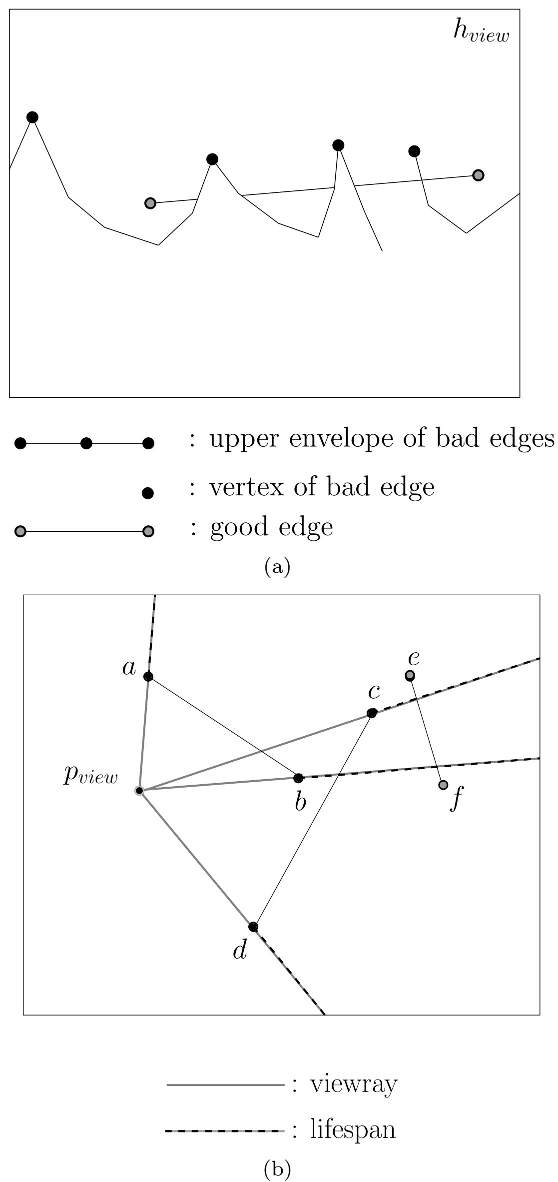


Figure 5.6 (a) A possible view of the upper envelope of the bad edges and a good edge at the back. (b) The xy-projection of a terrain that contains a good edge $\overline{ef}$ and two bad edges $\overline{ab}$, $\overline{cd}$. Illustrated are also the lifespans for the vertices of the bad edges.

Chapter 6

Conclusions and Future Research

In the present thesis we investigated various problems that have to do with the complexity of drainage and visibility structures on triangulated terrains. As with every work of this kind, solutions were provided for some of these problems, yet new questions arose from these solutions. Next we list the main contributions of this thesis together with open problems that can be the subject of future research.
In Chapter 2 we described an efficient mechanism that allows for computing information on drainage structures on TINs without explicitly computing the structures themselves. With this mechanism, for a terrain of n triangles, we can compute in $O(n \log n)$ time information such as the points where one or more flow paths end, or which triangles are fully included in the watersheds of one or more local minima. It remains an open problem if a similar mechanism can be designed to answer queries that are related to proximity structures on terrains, such as Voronoi diagrams. Also, it will be interesting to implement the presented algorithms and observe their performance in practice. Furthermore, the mechanism that we described assumes that no flat areas exist in the input terrain. This is actually the case for all the known algorithms that consider flow routing on TINs following strictly the direction of steepest descent. Thus, it is still an open problem to define a natural model for water flow on flat areas on TINs. Such models already exist for DEMs [81]. But so far there has been no attempt to translate this model for TINs, and examine the complexity of the induced drainage structures both in theory and practice.
In Chapter 3 we examined the problems that appear in practice when computing drainage structures on TINs using an exact flow model. We showed that it is inefficient to compute watershed subdivisions on large TIN data sets using a consistent flow model and exact arithmetic if the strip map is computed first. For this reason we described an output-sensitive algorithm that computes the watershed map

of a TIN based on the implicit mechanism of Chapter 2. It seems challenging to implement this algorithm and check if it is substantially more efficient than the algorithm that computes the strip map.
We also evaluated the quality of the output of several efficient heuristics for computing watersheds; our goal was to indicate which heuristic provides output that is closest to the exact watershed subdivision on the TIN. The results that the examined heuristics yielded were quite satisfactory for most experiments, yet there still seems to be some room for improvement, especially when computing watersheds on almost flat TINs. In any case it appears challenging to investigate thoroughly the conditions under which known heuristics fail to give a nice approximation and then, possibly, infer new methods that will improve the ouput quality.
As presented in Chapter 4, the existence of noise in the input data affects the drainage characteristics of the terrain. We showed how important it is to select the appropriate algorithmic method for identifying which parts of a TIN maintain their drainage properties when subject to noise in the vertex elevations. At this point, we should indicate another issue that is related to the use of digital terrain models in general (and not only TINs): a digital terrain model should be a good approximation of the real-world entity that it represents but this might not be always the case. The topologies of the original and the digital surface may differ significantly. Recall that topological structures such as the surface network of a terrain are based on the computation of flow paths on the terrain surface. Any discrepancies in the topology of a digital terrain and the original surface may be the result of technical issues related to the sampling process; errors in the physical measurements and undersampling. Yet, artefacts in the topology of the digital terrain may not only be due to noise in the sample point set but may also derive from the process itself of building the digital model out of this sample. At this stage, we should consider three possible factors that may produce artefacts during the reconstruction process:

- The first factor is the interpolation method that is employed to create extra points during the surface reconstruction process. This is very important in the case of DEMs, since there we have to compute elevations for the canonically spaced grid cells, while the points of the sample usually appear in an irregular pattern. Although a TIN can be built straightforwardly using the sample points as its vertex set, interpolation can be used to create extra vertices in order to enhance the approximation.

- Another factor is the geometry of the surface of the digital model; a TIN implies a piecewise-linear surface while a DEM is usually interpreted as a surface with many discontinuities. The critical points of the TIN are restricted to the vertex set of the triangulation, while usually for DEMs the critical points appear only in grid cell centers. An exception to this rule is provided by Steger [84]. Implicit surfaces are sometimes used to alleviate this restriction. When considering TINs, the triangulation method that is used to construct the terrain may also influence the quality of the approximation.

- The third factor is the flow model that is considered for representing the course that water would follow on the digital surface.

In this thesis, and especially in Chapter 3, we examined many issues that are related to flow models on TINs. Yet, the two first factors deserve more attention as well. Selecting an interpolation method, a digital terrain representation and a flow model is not a straightforward task; there are many choices for each of these three issues and consequently many more possible combinations. Thus, it becomes interesting to evaluate how these choices influence the creation of artefacts in the topology of the digital terrain representation. Given popular methods for reconstructing terrain surfaces, are there large differences when comparing the surface network and the watershed map on the reconstructed surface with the respective structures on the original terrain? Which methods produces the most faithful approximation? Of course, to answer this question we need a point of reference; we need to know the topology of the original surface. Thus, synthetic terrain data sets of known drainage properties can be used as the reference surfaces. Here it becomes important to use a precise flow model for computing the exact surface network and watershed map of the reference surface. The software package that we described in Chapter 3 allows for using piecewise-linear surfaces as a point of reference; with this software we can compute the exact watershed map and surface network of a TIN and thus we can use such a surface as a ground truth for testing the performance of any reconstruction method.
Finally, in Chapter 5 we considered an alternative approach for explaining why visibility maps of TINs do not have a high combinatorial complexity in practice. We used the concept of smoothed complexity to show that a visibility map of a TIN most likely has linear complexity with respect to the size of the TIN. It remains to find out if smoothed complexity can also be used to explain the low complexity of other structures on TINs such as flow paths or watersheds. In addition to proving a theoretical bound for the smoothed complexity of these structures, it would be interesting to investigate this problem experimentally. Considering TINs with drainage networks of high complexity, experiments may involve perturbing the TIN vertices and then evaluating any subsequent changes in the complexity of these networks.

References

[1] P. Alho and J. Aaltonen. Comparing a 1D Hydraulic Model with a 2D Hydraulic Model for the Simulation of Extreme Glacial Outburst Floods. *Hydrological Processes* 22(10):1537–1547 (2008).

[2] M. McAllister. A Watershed Algorithm for Triangulated Terrains. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 103–106, 1999.

[3] M. McAllister and J. Snoeyink. Extracting Consistent Watersheds from Digital River and Elevation Data. *Annual Conference of the American Society for Photogrammetry and Remote Sensing*, 1999.

[4] E.M. Arkin and R. Hassin On Local Search for Weighted k-Set Packing. *Mathematics of Operations Research* 23:640–648 (1998).

[5] B. Aronov, M. de Berg and S. Thite. The Complexity of Bisectors and Voronoi Diagrams on Realistic Terrains. In *Proc. 16th Annual European Symposium on Algorithms*, pages 100–111, 2008.

[6] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink and S. Yu. The Complexity of Rivers in Triangulated Terrains. In *Proc. 8th Canadian Conference on Computational Geometry*, pages 325–330, 1996.

[7] M. de Berg, O. Cheong, H. Haverkort, J.G. Lim and L. Toma. The Complexity of Flow on Fat Terrains and Its I/O-Efficient Computation. *Computational Geometry: Theory and Applications* 43:331–356 (2010).

[8] M. de Berg and K. Dobrindt. On Levels of Detail in Terrains. *Graphical Models and Image Processing* 60:1–12 (1998).

[9] M. de Berg, H. Haverkort and C.P. Tsirogiannis. Implicit Flow Routing on Terrains with Applications to Surface Networks and Drainage Structures. In *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 285–296, 2011.

[10] M. de Berg, H. Haverkort and C.P. Tsirogiannis. Visibility Maps of Realistic Terrains Have Linear Smoothed Complexity. *Journal of Computational Geometry* 1: 57–71 (2010).

[11] M. de Berg, A.F. van der Stappen, J. Vleugels, M.J. Katz. Realistic Input Models for Geometric Algorithms. *Algorithmica* 34:81–97 (2002).

[12] M. de Berg and C.P. Tsirogiannis. Exact and Approximate Computations of Watersheds on Triangulated Terrains. In *Proc. 19th ACM SIGSPATIAL Conference on Advances in Geographic Information Systems*, 2011. To appear.

[13] W. Bieniecki. Oversegmentation Avoidance in Watershed-Based Algorithms for Color Images. In *Proc. International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science*, pages 169–172, 2004.

[14] P. A. Brivio, R. Colombo, M. Maggi and R. Tomasoni. Integration of Remote Sensing Data and GIS for Accurate Mapping of Flooded Areas. *International Journal of Remote Sensing* 23(3):429–441 (2002).

[15] P.V. Bolstad, W. Swank, and J.Vose. Predicting Southern Appalachian Overstory Vegetation with Digital Terrain Data. *Landscape Ecology* 13:271–283 (1998).

[16] J.F. O'Callaghan and D.M. Mark. The Extraction of Drainage Networks from Digital Elevation Data. *Computer Vision, Graphics, and Image Processing* 28(3): 323–344 (1984).

[17] R. Carlson and A. Danner. Bridge Detection in Grid Terrains and Improved Drainage Enforcement. In *Proc. 18th ACM International Conference on Advances in Geographic Information Science (ACM GIS)*, pages 250–259, 2010.

[18] F. Cazals, F. Chazal and T. Lewiner. Molecular Shape Analysis Based upon The Morse-Smale Complex and The Connolly Function. In *Proc. 19th ACM Symposium on Computational Geometry*, pages 351–360, 2003.

[19] S. Chauduri and V. Koltun. Smoothed Analysis of Probabilistic Roadmaps. *Computational Geometry: Theory and Applications* 42:731–747 (2009).

[20] Z.-T. Chen and J. A. Guevara. Systematic Selection of Very Important Points (VIP) from Digital Terrain Model for Constructing Triangular Irregular Networks. In *Proc. 8th International Symposium on Computer-Assisted Cartography*, pages 50–56, 1987.

[21] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[22] C.M. Cobby, D. C. Mason, M. S. Horritt and P. D. Bates. Two-Dimensional Hydraulic Flood Modelling Using a Finite-Element Mesh Decomposed According to Vegetation and Topographic Features Derived from Airborne Scanning Laser Altimetry. *Hydrological Processes* 17(10):1979–2000 (2003).

[23] W. Collischonn, D.C. Buarque, A.R. da Paz, C.A.B. Mendes and F.M. Fan. Impact of Pit Removal Methods on DEM Derived Drainage Lines in Flat Regions. In *Proc. American Water Resources Association (AWRA) Spring Specialty Conference 2010*.

[24] L. Čomić, L. De Floriani and L. Papaleo. Morse-Smale Decompositions for Modeling Terrain Knowledge. In *Proc. 7th International Conference on Spatial Information Theory*, pages 426–444, 2005.

[25] M.C. Costa-Cabral and S.J. Burges. Digital Elevation Model Networks (DEMON): A Model of Flow Over Hillslopes for Computation of Contributing and Dispersal Areas. *Water Resources Research* 30(6):1681–1692 (1994).

[26] V. Damerow, F. Meyer auf der Heide, H. Räcke, C. Scheideler and C. Sohler. Smoothed Motion Complexity. In *Proc. 11th Annual European Symposium on Algorithms*, pages 161–171, 2003.

[27] V. Damerow and C. Sohler. Extreme Points Under Random Noise. In *Proc. 12th Annual European Symposium of Algorithms*, pages 264–274, 2004.

[28] A. Danner, T. Mølhave, K. Yi, P.K. Agarwal, L. Arge and H. Mitasova. TerraStream: From Elevation Data to Watershed Hierarchies. In *Proc. 15th ACM International Symposium on Advances in Geographic Information Science (ACM GIS)*, pages 212–219, 2007.

[29] A. Driemel, H. Haverkort, M. Loffler and R. Silveira. Flow Computations on Imprecise Terrains. In *Proc. 12th Algorithms and Data Structures Symposium (WADS)*, 2011. To appear.

[30] B. Ducke and P.C. Kroefges. From Points to Areas: Constructing Territories from Archaeological Site Patterns Using an Enhanced Xtent Model. In *Proc. of the 35th International Conference on Computer Applications and Quantitative Methods in Archaeology (CAA)*, pages 245–251, 2007.

[31] H. Edelsbrunner, J. Harer and A. Zomorodian. Hierarchical Morse–Smale Complexes for Piecewise Linear 2-Manifolds. *Discrete & Computational Geometry* 30(1):87–107 (2003).

[32] H. Edelsbrunner, D. Letscher and A. Zomorodian. Topological Persistence and Simplification. In *Proc. 41st IEEE Symposium on Foundations of Computer Science*, pages 45–463, 2000.

[33] T.A. Endreny and E.F. Wood. Representing Elevation Uncertainty in Runoff Modelling and Flowpath Mapping. *Hydrological Processes* 15:2223–2236 (2001).

[34] A. Fabri, F. Cacciola and R. Wein. CGAL and the Boost Graph Library. In *CGAL User and Reference Manual*, CGAL Editorial Board, 3.7 edition, 2010.

[35] J. Fairfield and P. Leymarie. Drainage Networks From Grid Digital Elevation Models. *Water Resources Research* 27(5):709–717 (1991).

[36] A. Frank, B. Palmer and V. Robinson. Formal Methods for the Accurate Definition of Some Fundamental Terms in Physical Geography. In *Proc. 2nd International Symposium Spatial Data Handling*, pages 585–599, 1986.

[37] P.F. Fisher. Extending the Applicability of Viewsheds in Landscape Planning. *Photogrammetric Engineering and Remote Sensing (PERS)* 62(11):1297–1302 (1996).

[38] T. Granlund. GMP, the GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`.

[39] C. Gray, F. Kammer, M. Löffler and R. I. Silveira. Removing Local Extrema from Imprecise Terrains . In *Abstracts of 26th European Workshop on Computational Geometry (EuroCG)*, pages 181–184, 2010.

[40] F. Gygax. *Das Topographische Relief in der Schweiz. Ein Beitrag zur Geschichte der Schweizerischen Kartographie.* Buchdruckerei Neukomm and Schalcrath, Bern, 1937.

[41] H. Haverkort and C.P. Tsirogiannis. Flow on Noisy Terrains: An Experimental Evaluation. In *Proc. 19th ACM SIGSPATIAL Conference on Advances in Geographic Information Systems*, 2011. To appear.

[42] T. Hazel, L. Toma, J. Vahrenhold and R. Wickremesinghe. Terracost: Computing Least-Cost-Path Surfaces for Massive Grid Terrains. In *Proc. of ACM Journal of Experimental Algorithmics* 12:Article 1.9, 2008.

[43] F. Hebeler and R.S. Purves. The Influence of Elevation Uncertainty on Derivation of Topographic Indices. *Geomorphology* 111(1-2):4–16 (2009).

[44] W.E. Higgins and E.J. Ojard. Interactive Morphological Watershed Analysis for 3D Medical Images. *Computer Medical Imaging and Graphics* 17 (4 – 5) : 387 – 95 (1993).

[45] M. Hemmer, S. Hert, L. Kettner, S. Pion and S. Schirra. Number Types. In *CGAL User and Reference Manual.* CGAL Editorial Board, 3.8 edition, 2011.

[46] S. Jenson and J. Domingue. Extracting Topographic Structures from Digital Elevation Data for Geographic Information System Analysis. *Photogrammetric Engineering and Remote Sensing* 54(11):1593–1600 (1988).

[47] N. L. Jones, S. G. Wright and D. R. Maidment. Watershed Delineation with Triangle-Based Terrain Models. *Journal of Hydraulic Engineering* 116(10):1232–1251 (1990).

[48] M.J. Katz, M.H. Overmars and M. Sharir. Efficient Hidden Surface Removal for Objects With Small Union Size. In *Proc. 7th ACM Symposium on Computational Geometry (SOCG)*, pages 31–40, 1991.

[49] S.J. Kay and T. Sly. An Application of Cumulative Viewshed Analysis to a Medieval Archaeological Study: The Beacon System of the Isle of Wight, United Kingdom. *Archeologia e Calcolatori* 12:167–179 (2001).

[50] S.J. Kay and R.E. Witcher Predictive Modelling of Roman Settlement in the Middle Tiber Valley. *Archeologia e Calcolatori* 20:277–290 (2009).

[51] M.W. Lake and P.E. Woodman. Visibility Studies in Archaeology: a Review and Case Study. *Environment and Planning B: Planning and Design* 30(5):689–707 (2003).

[52] M. Llobera. Building Past Landscape Perception With GIS: Understanding Topographic Prominence. *Journal of Archaeological Science* 28(9):1005–1014 (2001).

[53] M. Lanthier, A. Maheshwari and J-R. Sack. Approximating Weighted Shortest Paths on Polyhedral Surfaces. In *Proc. 13th ACM Symposium on Computational geometry (SOCG)*, pages 274–283, 1997.

[54] M. Lanthier, A. Maheshwari and J-R. Sack. Shortest Anisotropic Paths on Terrains. In *Proc. 26th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 524–533, 1999.

[55] D. Letscher and J. Fritts. Image Segmentation Using Topological Persistence. In *Proc. 12th International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 587–595, 2007.

[56] J.B. Lindsay and M.G. Evans. The Influence of Elevation Error on the Morphometrics of Channel Networks Extracted From DEMs and the Implications for Hydrological Modelling. *Hydrological Processes* 22(11):1588–1603 (2008).

[57] Y. Liu and J. Snoeyink. Flooding Triangulated Terrain. In *Proc. 11th International Symposium on Spatial Data Handling*, pages 137–148, 2005.

[58] S. Mackay and L. Band. Extraction and Representation of Nested Catchment Areas from Digital Elevation Models in Lake-Dominated Topography. *Water Resources Research Journal* 34(4):897–901 (1998).

[59] D.R. Maidment. Developing a Spatially Distributed Unit Hydrograph by Using GIS. In *Proc. HydroGIS 93*, pages 181–192, 1993.

[60] A. Mangan and R. Whitaker. Partitioning 3D Surface Meshes Using Watershed Segmentation. *IEEE Transaction on Visualization and Computer Graphics* 5(4):308–321 (1999).

[61] J. Milnor. *Morse Theory.* Princeton University Press, New Jersey, 1963.

[62] H. Mitasova and J. Hofierka. Interpolation by Regularized Spline with Tension: II. Application to Terrain Modeling and Surface Geometry Analysis. *Mathematical Geology* 25(6):657–667 (1993).

[63] J.S.B. Mitchel, D. S. Mount and C.H. Papadimitriou. The Discrete Geodesic Problem. *SIAM Journal on Computing* 16(4):647–668 (1987).

[64] E. Moet. *Computation and Complexity of Visibility in Geometric Environments.* PhD Thesis, Utrecht University, 2008.

[65] E. Moet.

Experimental Verification of a Realistic Input Model for Polyhedral Terrains. Technical report UU-CS-2007-052, Utrecht University, 2007.

[66] E. Moet, M. van Kreveld and A.F. van der Stappen. On Realistic Terrains. *Computational Geometry: Theory and Applications* 41:48–67 (2008).

[67] P. Moreels and S. Smrekar. Watershed Identification of Polygonal Patterns in Noisy SAR Images. *IEEE Transactions on Image Processing* 12:740–750 (2003).

[68] O. Palacios-Velez and B. Cuevas-Renaud. Automated River-Course, Ridge and Basin Delineation from Digital Elevation Data. *Journal of Hydrology* 86:299–314 (1986).

[69] D. Papa and I. Markov. Hypergraph Partitioning and Clustering. In: T. Gonzalez(ed.), *Approximation Algorithms and Metaheuristics* . CRC Press, pages 6-1–6-19, 2007.

[70] J. Pfaltz. Surface Networks. *Geographical Analysis Journal* 8:77–93 (1976).

[71] M. Revsbæk. *I/O Efficient Algorithms for Batched Union-Find with Dynamic Set Properties and its Application to Hydrological Conditioning.* Master's Thesis, Department of Computer Science, Aarhus University, 2007.

[72] P. Quinn, K. Beven, P. Chevallier and O. Planchon. The Prediction of Hillslope Flow Paths for Distributed Hydrological Modelling Using Digital Terrain Models. *Hydrological Processes* 5(1):59–79 (1991).

[73] S. Rana and J. Morley. Application of Surface Networks for Fast Approximation of Visibility Dominance in Mountainous Terrains. In: S. Rana (ed.), *Topological Data Structures for Surfaces*, pages 167–176, 2004.

[74] N. J. Lea. An Aspect-Driven Kinematic Routing Algorithm. In: A. J. Parson and A. D. Abrahams (eds.), *OverLand Flow: Hydraulics and Erosion Mechanics*, UCL Press, pages 393–408, 1992.

[75] D. Rippin, I. Willis, N. Arnold, A. Hodson, J. Moore, J. Kohler and H. Björnsson. Changes in Geometry and Subglacial Drainage of Midre Lovènbreen, Svalbard, Determined from Digital Elevation Models. *Earth Surface Processes and Landforms, Special Issue: The Generation of High Quality Topographic Data for Hydrology and Geomorphology* 28(3):273–298 (2003).

[76] J.K. Rød and D. van der Meer. Visibility and Dominance Analysis: Assessing a High-Rise Building Project in Trondheim. *Environment and Planning B: Planning and Design* 36(4):698–710 (2009).

[77] R.L. Saunders. *Terrainosaurus: Realistic Terrain Synthesis Using Genetic Algorithms.* Master's Thesis, Texas A& M University, 2006.

[78] Bernhard Schneider. Extraction of Hierarchical Surface Networks from Bilinear Surface Patches. *Geographical Analysis* 37:244–263 (2005).

[79] M. Sharp, K. Richards, I. Willis, N. Arnold, P. Nienow, W. Lawson and J-L. Tison. Geometry, Bed Topography and Drainage System Structure of the Haut Glacier d'Arolla, Switzerland. *Earth Surface Processes and Landforms* 18(6):557–571 (1993).

[80] C. Smemoe, J. Nelson and A. Zundel. Risk Analysis Using Spatial Data in Flood Damage Reduction Studies. In *Proc. World Water and Environmental Resources Congress*, 2004.

[81] P. Soille, J. Vogt and R. Colombo. Carving and Adaptive Drainage Enforcement of Grid Digital Elevation Models. *Water Resources Research* 39(12):1366–1375 (2003).

[82] D.A. Spielman and S.H. Teng. Smoothed Analysis of Algorithms: Why The Simplex Algorithm Usually Takes Polynomial Time? *Journal of the ACM* 51:385–463 (2004).

[83] C.W. Stahl. *Accumulated Surfaces & Least-Cost Paths: GIS Modeling for Autonomous Ground Vehicle (AGV) Navigation.* Master's Thesis, Virginia Polytechnic Institute and State University, 2005.

[84] C. Steger. Subpixel-Precise Extraction of Watersheds. In *Proc. the Seventh IEEE International Conference on Computer Vision*, pages 884–890 (vol. 2), 1999.

[85] D. C. Stempien. Terrain Models as Battlefield Visualization Training Tools. *Military Intelligence Professional Bulletin* 28(4):33 (2002).

[86] S.L. Stoev and W. Straer. Extracting Regions of Interest Applying Local Watershed Transformation. In *Proc. IEEE Visualization*, pages 21–29, 2000.

[87] S. Takahashi, T. Ikeda, T.L. Kunii and M. Ueda. Algorithms for Extracting Correct Critical Points and Constructing Topological Graphs from Discrete Geographic Elevation Data. *Computer Graphics Forum* 14(3):181–192 (1995).

[88] D.G. Tarboton. A New Method for the Determination of Flow Directions and Contributing Areas in Grid Digital Elevation Models. *Water Resources Research* 33(2):309–319 (1997).

[89] D. Theobald and M. Goodchild. Artifacts of TIN-Based Surface Flow Modeling. In *Proc. of GIS/LIS'90*, pages 955–964, 1990.

[90] United States Geological Survey, Seamless Data Warehouse Webpage. `http://seamless.usgs.gov/`.

[91] United States Geological Survey DEM Data Set Repository. `http://dds.cr.usgs.gov/pub/data/DEM/250/`.

[92] L. Vincent and P. Soile. Watershed in Digital Spaces: An Efficient Algorithm Based on Immersion Simulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13(6):583–598 (1991).

[93] L.T. Watson, T.J. Laffey and R.M. Haralick. Topographic Classification of Digital Image Intensity Surfaces Using Generalized Splines and the Discrete Cosine Transformation. *Computer Vision, Graphics and Image Processing* 29(2):143–167 (1985).

[94] S.Wechsler. Uncertainties Associated with Digital Elevation Models for Hydrologic Applications: A Review. *Hydrology and Earth System Science* 11(4):1481–1500 (2007).

[95] M.F. Worboys and M. Duckham. *GIS: A Computing Perspective, Second Edition*, CRC Press, 2004.

[96] S. Yu, M. van Kreveld and J. Snoeyink. Drainage Queries in TINs: From Local to Global and Back Again. In *Proc. 7th International Symposium on Spatial Data Handling*, pages 13A.1–13A.14, 1996.

[97] X. Zhu, R. Sarkar and J. Gao. Topological Data Processing for Distributed Sensor Networks with Morse-Smale Decomposition. In *Proc. 28th Annual IEEE Conference on Computer Communications (INFOCOM09), mini-conference*, pages 2911–2915, 2009.

Summary

Landscapes and their morphology have been widely studied for predicting physical phenomena, such as floods or erosion, but also for planning human activities effectively, such as building prominent fortifications and watchtowers. Nowadays, the study of terrains is done in a computer-based environment; terrains are modelled by digital representations, and algorithms are used to simulate physical processes like water flow and to compute attributes like visibility from certain locations. In the current thesis we focus on designing new algorithms for computing structures related to water flow and visibility on digital terrain representations. Most specifically, the terrain representations that we considered are the so-called Triangulated Irregular Networks (TINs), that is, piecewise linear surfaces that consist of triangles.

One of the problems that are considered is the effect of noise on the worst-case complexity of visibility structures on TINs. The view that a person can have from a point on the surface of a TIN can be very complex, since in the worst case thin obstacles in the foreground may appear to fragment many long terrain edges in the background into visible and invisible pieces. In our analysis we considered TINs whose triangles have some well-defined properties that terrains in practice are expected to have. Although complex visibility structures can be induced on such TINs as well, we proved formally that slight perturbations on the elevations of the TIN vertex set will always get rid of the high complexity.

Another key problem that is studied is to design efficient algorithms that compute flow-related structures on TINs. So far it was known that, in the case of TINs, drainage structures that were computed using a consistent flow-model could have high complexity for specific input instances. We managed to develop a mechanism that can extract important information on flow paths and other drainage structures without computing those structures explicitly. This mechanism can be used as a basis for designing a variety of efficient algorithms, such as for computing the area measure of drainage structures or for computing structures that represent the terrain topology.

The last part of the presented work involves the implementation of a software package that computes drainage structures on TINs. In this package flow is modelled as following strictly the direction of steepest descent on the TIN surface. Existing software for related applications either constrain flow on the edge set of the TIN, or use inexact arithmetic, both of which introduces imprecise and/or incorrect results in the output. Our implementation is the first one that, at the same time, follows a robust flow model and uses exact arithmetic. We have used this implementation as a point of reference for evaluating experimentally the quality of the output of other flow models which are used in many hydrological applications. We have also used our software for conducting experiments on extracting watersheds on imprecise TINs, that is, TINs where the elevation values of the vertices are not exactly defined but are subject to noise from some given interval. Based on the results of these experiments, we have designed a novel method for extracting watersheds on imprecise terrains that produces high quality output.

Curriculum Vitae

Constantinos Tsirogiannis was born on the November 4th, 1983 in Komotini, Greece. He graduated from the 3rd Lyceum of the city of Volos, Greece in 2001. He received his Bachelor of Science in Informatics and Telecommunications from the National and Kapodistrian University of Athens in 2006. In 2007, he completed in the same department his Master of Science Degree in Computational Science and Algorithms. Since November 2007, he has been a Ph.D. student within the department of Mathematics and Computer Science of the Technische Universiteit Eindhoven (TU/e).

Titles in the IPA Dissertation Series since 2005

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization.* Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML.* Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models.* Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications.* Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition.* Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language*

Conglomerates. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation.* Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time.* Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Trans-*

formation. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*.

Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of

Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16